الجمهورية العربية السورية
وزارة التعليم العالي
جامعة دمشق



خوارزميات المعالجة المتوازية وبرمجتها

دراسة أعدت لنيل شهادة الماجستير في المعلوماتية

إعداد

كندة زين العابدين

المشرف الأساسي
الأستاذ الدكتور
خالد علي الخنيفس

المشرف المساعد
الدكتور
حامد خالد الرجوب

١٣ / ٩ / ٢٠٠٦
٢١ شعبان ١٤٢٧



* وسَارِعُوا إلى مَغفرَةٍ من ربِّكُم وجَنَّةٍ عَرضُهَا السَّمَاوَاتُ والأَرضُ أُعِدّتْ للمُتّقين *

****

*** كلمة شكر ***

لا يسعني في مقدمة هذه الرسالة إلا أن أشكر
الأستاذ الدكتور خالد علي الخنيفس الذي تفضل
مشكوراً بالإشراف على هذا العمل وكان عوناً لي
على إنجازه وإخراجه بهذه الصورة كما أشكر
الدكتور حامد خالد الرجوب الذي ساهم في
الإشراف على هذه الرسالة

****

الفهرس

خوارزميات

المعالجة المتوازية

وبرمجتها

مقدمة

تطور علم المعلوماتية منذ بدايته في منتصف القرن العشرين حتى بداية هذا القرن بشكل متسارع وكبير وتشعبت فروعه وتعددت مجالاته واختصاصاته. وأصبح دخول هذا العلم ضرورياً في مختلف مجالات الحياة وكافة فروع العلوم الأخرى واقتضى ذلك تصميم الخوارزميات الملائمة لحل المسائل والمشكلات المطروحة إلى تطوير كبير في علم الخوارزميات. واقتضت بنية الحواسيب التي رافق ظهورها ولادة علم المعلوماتية إلى وضع خوارزميات تنفذ تتابعياً بحيث تكون تعليمات البرامج مرتبة ترتيباً دقيقاً ويتم تنفيذها الواحدة تلو الأخرى، إلا أنه مع زيادة حجم المسائل الواجب حلها وتشعب المشكلات الواقعية المطروحة وصعوبتها أصبح من العسير على مثل هذه الخوارزميات الوصول لحل دقيق لتلك المسائل والمشكلات إما لتجاوز قدرات الذاكرات اللازمة لتخزين المعلومات والنتائج المرحلية أو بسبب المدد الزمنية الكبيرة اللازمة للوصول للحل الأمثل وعلاوة على ذلك ازداد إلحاح مستخدمي الحواسيب في طلب قدرات حسابية متنامية وذلك بشكل مستمر الأمر الذي حدا بالعلماء والباحثين للتفكير في إيجاد أساليب جديدة تساعد على تلبية تلك الرغبات سواء في تسريع عملية البحث عن حلول للمسائل المطروحة أو زيادة حجم تلك المسائل.

ومن هنا ولدت مسألة التوازي في البرمجة إلا أنه تبين استحالة تطبيقها على الأجهزة التقليدية التي تعتمد تقنية الكترونية ثابتة ، الأمر الذي استدعى التدخل في بنية الجهاز نفسه والبحث عن بنية جديدة وقد تم في مطلع الثمانينات من القرن العشرين تصميم أول حاسوب متعدد المعالجات دعي CRAY1 الذي احتوى على معالجين يعملان في آن معاً وتوالى بعد ذلك ظهور حواسيب جديدة تعتمد بنى مختلفة مثل DAP, BSP, Illiaclv, Cyber 205, ... وكان القاسم المشترك بين مختلف هذه البنى هو استخدام عدة معالجات في الحاسوب الواحد، وعلى الرغم من الإمكانات الهائلة التي تقدمها هذه الأجهزة سواء في قدرتها الكبيرة في الحساب أو في حجم الذاكرات الكبير جداً، إلا أن ظهورها قد طرح مشكلة أساسية هي عدم المحافظة على التسلسل الزمني لتنفيذ التعليمات والعمليات مما استدعى تطوير خوارزميات دعيت بالمتوازية لتتلاءم مع الواقع الجديد

الذي فرضته هذه الحواسيب فظهرت خوارزميات متوازية لكـل مـن الأنـواع العديـدة للحواسيب المتوازية.

ونظراً لأهمية هذا الموضوع في التقدم الحاصل في استخدام المعلوماتيـة بوصـفها الأداة المفضلة في كافة المجالات فقد رغبت بالمساهمة في دراسته لعلي أقدم مساهمة آمـل أن تكون ذات فائدة على كافة الأصعدة.

ويمكن اعتبار الأهداف الرئيسية لهذا العمل:

1- عرض مفاهيم الخوارزميات المتوازية.

2- تطوير مفهوم الخوارزميات والمعالجة المتوازية وتطوير إمكانية تطبيق الخوارزميات المتوازية في معالجة الأنظمة التطبيقية.

تقع الرسالة في ثلاثة فصول:

- خصص الفصل الأول لدراسة مفهوم الأنظمة متعددة المعالجات وتعريفها وتصنيفها ودراسة أسلوب التواصل فيما بينها، ويتضمن هذا الفصل تعريف تعدديـة المهـام وتعريف الأنظمة متعددة المعالجات ومن ثم وصفها وتصنيفها، وبعد التعرف على أنواع الحواسيب تبعاً للتعليمات وتحكمها بالبيانات نتعرف على أسـلوب الاتصـال بين وحدات المعالجة في هذه الأنظمة ونتحدث عنها بالتفصيل. كما أننا نقدم أفكاراً ونتائج تشكل أساسات نظرية الحساب المتوازي ونوضح بعض نمـاذج الحسـاب المتوازي. كما نتحدث عن الدارات المنطقية التي تعتبر نموذجـاً مصـغراً عـن الحساب المتوازي وتعتبر مقياساً لكلفة التوازي فهي تشكل قاعدة لدراسـة تعقيـد الحسابات المتوازية.
- خصص الفصل الثاني لدراسة الخوارزميات المتوازية وأنواعها وتمييز خواصـها وتقنيات تصميمها ومقاييس تحليلها وتحديد كفاءتها، ويتضمن هذا الفصـل مقدمـة عن الخوارزميات المتوازية وتعريفاً لها ومن ثم تحليل الخوارزميـات المتوازيـة وتحديد المعايير الخاصة بذلك وبعدها نقوم بشرح بعضـاً مـن تقنيـات تصـميم الخوارزميات المتوازية والتسلسلية وأخيراً نحدد خواص الخوارزميات المتوازية.
- خصص الفصل الثالث لدراسة خوارزميات متوازية لحل بعـض المسـائل علـى أجهزة متعددة المعالجات، ويتضمن هذا الفصل شرحاً مفصلاً لخوارزميات متوازية

على نماذج مختلفة من الحواسيب متعددة المعالجات ومن ثم تحليلها وذكر خواصها ومقارنتها بالخوارزميات التسلسلية للمسألة. إضافة إلى عرض بعض تطبيقات الخوارزميات المتوازية مثل مسألة حقيبة الظهر ومسألة المجاميع البادئة ومسألة الأعمال التسلسلية وفق فترات انتهائها كما نبين فيما بعد الخوارزميات المختلفة لمسائل رياضية شهيرة على أشكال مختلفة من الأجهزة متعددة المعالجات ونذكر الأمثلة التوضيحية الموافقة لكل منها.

كما تحتوي الرسالة على ملحق يتضمن البرامج الحاسوبية لبعض الخوارزميات إضافة لعرض بعض الأمثلة التوضيحية لعملها.

وقد توخيت البساطة في العرض مع المحافظة على الدقة العلمية، كما أني أسقطت إثبات بعض النظريات التي لا مناص من ذكرها حفاظاً على التسلسل المنطقي في عرض المادة العلمية وفهمها.

وإني أرجو الله أن أكون قد وفقت في معالجة هذا الموضوع وقدمت مساهمة بسيطة في تطوير جانباً مهماً من جوانب علم المعلوماتية.

وأسأل الله العون إنه نعم المولى ونعم النصير.

الفصل الأول

دراسة مفهوم الأنظمة متعددة المعالجات وتعريفها وتصنيفها ودراسة أسلوب التواصل فيما بينها

1-1 مفهوم البرمجة متعددة المهام:

1-1-1 تعريف النظم المعتمدة على الحاسوب:

تعرف النظم المعتمدة على الحاسوب بأنها مجموعة أو ترتيب عناصر نُظمت بغية إنجاز هدف محـــدد بمعالجة المعلومات. ويستخدم النظام المعتمد على الحاسوب عدداً من العناصـــر وهـــي: البرمجيـــات والعتاديات والأشخاص وقاعدة المعطيات والوثائق والإجراءات. وتُضم هذه العناصر بأشكال مختلفـــة لتحويل المعلومات.

ولإنشاء نماذج لهذه النظم نقوم بما يلي:

- تعريف الإجرائيات التي تخدم احتياجات الرؤية المعينة
- تمثيل سلوك الإجرائيات والافتراضات التي يستند إليها هذا السلوك
- تعريف الدخل الخارجي والدخل الداخلي للنموذج بشكل واضح
- تمثيل جميع الروابط (بما فيها الخرج) التي تساعد المهندس على فهم الرؤية فهماً أفضل.

ولبناء نموذج نظام يجب على المهندس التقيد بالعوامل التالية:

1- الافتراضات: التي تخفض عدد التبادلات والاختلافات الممكنة والتي تسمح من ثم للنموذج أن يعكس المشكلة على وجه مقبول.
2- التبسيطات: التي تسمح بإنشاء النموذج بالوقت المناسب.
3- الحدود: التي تساعد في وضع حدود للنظام.
4- القيود: التي توجه الطريقة التي يُنشأ بها النموذج والمنهج المتبع في تنجيزه.
5- التفضيلات التي تدل على البنيانات المفضلة لجميع المعطيات والوظائف والتقانة.

قد يترتب على نموذج النظام الناتج حل تام الأتمتة أو نصف مؤتمت أو يدوي. ويمكن وصف نمـــاذج لكل نمط يحقق حلاً بديلاً للمسألة المطروحة وببساطة يتم تعديل التأثير النسبي لعناصر النظام المختلفة (الأشخاص والعتاديات والبرمجيات) بغية استنتاج نماذج من كل نمط.

2-1-1 تعريف تعددية المهام:

تعرف تعددية المهام (Multitasking)على أنها قدرة نظام التشغيل على تنفيذ أكثر من مهمة واحـدة في الوقت نفسه وتتيح تعددية المهام مثلاً تنفيذ عدة برامج مختلفة والطباعة على الطابعـة والتخـزين على القرص في الوقت نفسه ودون انتظار انتهاء كل مهمة من هذه المهمات قبل تنفيذ الأخرى.

وتبنى الخوارزمية في الأجهزة المتعددة المعالجات على أساس تعددية المهام والتنفيذ المتزامن حيـث تتولى كل وحدة معالجة مسألة جزئية فتقوم بمعالجتها بشكل متزامن مع الوحدات الأخرى ونحصل في النهاية على النتيجة بجمع تلك النتائج المتعددة والحصول على الحل النهائي للمسألة. وتعتمد البرمجـة المتعددة المهام على فهم طريقة تنفيذ التعليمات على البيانات حيث إن هناك سلسلة من التعليمات تأمر الحاسب بوظيفته في كل خطوة وهناك سلسلة من البيانات (دخل الخوارزمية) ستتأثر بتلك التعليمات.

ويتم تصنيف الحواسيب تبعاً لطريقة تنفيذ التعليمات على البيانات وتتم برمجة كل نـوع مـن أنـواع الحواسيب باستخدام خوارزمية إما تسلسلية وإما متوازية كل على حسب ما يوفر من إمكانيات. كما أنه من الضروري وجود اتصال بين وحدات المعالجة بعضها مع بعض خلال الحساب وذلك مـن أجـل تغيير البيانات أو النتائج ويتحقق هذا الاتصال عن طريق الذاكرة المشتركة للنظام أو عن طريق شبكة الاتصالات بين وحدات المعالجة.

2-1 الأنظمة المتعددة المعالجات:

1-2-1 وصف الأنظمة المتعددة المعالجات:

لقد وجدت الأنظمة المتعددة المعالجات من أجل تحسين الفعاليات وذلك مقارنـة بالكلفـة والموثوقيـة والتكيّف ويعتبر النظام المتعدد المعالجات (مع أو بدون توازٍ) رائداً بين أنظمة المعلوماتية حيث ينافس نظام المعالج الوحيد وينافس أيضاً الأنظمة الموزعة وغيرها.

وفي النظام المتعدد المعالجات المتوازي يوجد العديد من وحدات المعالجة المسـتقلة والتـي تتصـف بذاكرات محلية وتشترك معاً بذاكرة رئيسية وتكون هذه المعالجات متزامنة. حيث يحـوي الحاسـب المتوازي عدة معالجات ويقوم بإعطاء مسألة جزئية من المسألة الكلية لكل منها وتحل المسائل الجزئية بشكل متزامن كل واحدة في معالج مختلف ومن ثمّ تجمع النتائج لإعطاء حل المسألة الأصلية.

يتكون أي حاسب سواء تسلسلي أو متوازٍ من تعليمات تنفذ على البيانات: حيث تأمر سلسلة من التعليمات الحاسب بما سيقوم به في كل خطوة ، وتتأثر سلسلة بيانات (وهي دخل الخوارزمية) بتلك التعليمات.

نستطيع أن نميز بين أربعة أنواع من الحواسيب:

1- أولاً: حواسيب SISD (سلسلة تعليمات مفردة، سلسلة بيانات مفردة):

Single Instruction stream, Single Data stream(SISD)

وهي حواسيب تعلمية – تعليمية حيث يتكون الحاسب في هذه الفئة من وحدة معالجة مفردة تتلقى سلسلة تعليمات مفردة. تصدر وحدة التحكم تعليمة واحدة تطبقها على عنصر البيانات الذي تحصل عليه من وحدة الذاكرة. نستخدم لبرمجة حاسب من هذه الفئة خوارزمية تسلسلية أو تتابعية. [SeGA-1989]

2- ثانياً: حواسيب MISD (سلسلة تعليمات متعددة – سلسلة بيانات مفردة):

Multiple Instruction stream, Single Data stream (MISD)

يوجد في هذا النوع من الحواسيب N معالجاً كل منها يملك وحدة تحكم خاصة به ويتقاسم وحدة الذاكرة المشتركة حيث توجد البيانات. يصل في كل خطوة عنصر واحد من البيانات من الذاكرة ويتم العمل به من قبل جميع المعالجات بآن واحد كلٌ حسب التعليمة التي وصلت إليه من وحدة تحكمه الخاصة. يتحقق التوازي هنا بجعل المعالجات تعمل عدة أشياء مختلفة في الوقت نفسه على عنصر البيانات نفسه وهذا النوع من الحواسيب يساعد نفسه بشكل طبيعي من أجل الحسابات التي تتطلب إدخالاً ليصبح مادة لعمليات متعددة فيتلقى كل منها الإدخال بالشكل الأصلي.[SeGA-1989]

مثال1-1:

لتكن لدينا مسألة تحديد ما إذا كان عدد معطى z أولياً أو لا.

إن الحل المعروف لهذه المسألة هو تجريب جميع احتمالات قاسم العدد Z : فإذا لم ينجح أي منها في القسمة على Z عندها يكون أولياً وإلا فإنه يكون مركباً أو غير أولي.

تكمن الفكرة هنا في تقسيم العمل في اختبار القواسم الممكنة بين المعالجات. حيث تأخذ كل المعالجات Z على أنه إدخال ومن ثم يحاول كل منها تقسيم العدد على القواسم المحتملة المساعدة الموجودة عنده

ويناقش قاعدة الخرج المخصصة على النتيجة. وهكذا يمكن تحديد ما إذا كان Z أولياً بخطوة واحدة. أما إذا كان لدينا عدد قليل من المعالجات أقل من القواسم المحتملة فإن كل من المعالجات يستطيع إنجاز العمل في اختبار مجموعة فرعية مختلفة من القواسم.

3- ثالثاً: حواسيب SIMD (سلسلة تعليمات مفردة – سلسلة بيانات متعددة):

Single Instruction stream, Multiple Data stream (SIMD)

يتكون الحاسب المتوازي في هذه الفئة من N معالجاً متماثلاً كل واحد منها يمتلك ذاكرته المحلية الخاصة والتي يمكن أن يخزن فيها البرنامج والبيانات. تعمل كل المعالجات تحت تحكم سلسلة تعليمات مفردة تمت معالجتها في وحدة التحكم المركزية. وبشكل مكافئ فإنه من المفترض أن تمتلك كل من الـ N معالج نسخاً متطابقة من البرنامج المفرد. وتخزن كل نسخة برنامج في الذاكرة المحلية لكل معالج وهناك أيضاً لكل معالج N سلسلة بيانات. تعمل المعالجات بشكل متزامن ، حيث تنفذ كل المعالجات التعليمة نفسها في كل خطوة ويمكن أن تكون التعليمة بسيطة (مثل جمع أو طرح عددين) أو تعليمة معقدة مركبة (مثل دمج قائمتين من الأرقام) وبشكل مشابه يمكن أن يكون عنصر البيانات بسيطاً (رقماً واحداً) أو مركباً (عدة أرقام).

ربما يكون ضرورياً في بعض الأحيان أن نملك مجموعة فرعية فقط من المعالجات تقوم بتنفيذ تعليمة ما ويمكن أن نرمز هذه المعلومات في التعليمة نفسها فنخبر المعالج ما إذا كان عليه أن يكون فعالاً (وينفذ التعليمة) أو غير فعال (وينتظر من أجل أن يتلقى التعليمة التالية).

في معظم المسائل الهامة التي نحلها باستخدام حواسيب من هذا النوع يكون من المستحب أن تكون المعالجات قادرة على الاتصال مع بعضها خلال الحساب من أجل تغيير البيانات أو النتائج المكررة. ونحقق هذا الاتصال وفقاً لفئتين فرعيتين: إحداهما يكون الاتصال فيها عن طريق الذاكرة المشتركة والأخرى عن طريق شبكة الاتصالات.

(1-3) حواسيب SIMD (SM) بذاكرة مشتركة Shared-Memory :

تعرف هذه الفئة أيضاً بآلة الوصول العشوائي المتوازي نموذج (PRAM) The Parallel Random Access Machine .

يتم تقاسم الذاكرة المشتركة هنا والتي يستخدمها مجموعة من الناس بنفس الطريقة ، عندما يريد معالجان الاتصال فإنهما يعملان خلال الذاكرة المشتركة . لنفترض أن المعالج I يرغب بتمرير قيمة للمعالج ز يتم

هذا بخطوتين: الأولى يكتب المعالج I الرقم في قسم الذاكرة في موقع معروف من قبل المعـــالج j والـــذي يقرؤه بدوره من ذلك الموقع.

خلال تنفيذ خوارزمية التوازي فإن الـــ N معالج تصل إلى الذاكرة المشتركة لقراءة البيانات المدخلة مـــن أجل قراءة أو كتابة النتائج المكررة ومن أجل كتابة النتائج النهائية. يســـمح النمـــوذج الأساســـي لجميـــع المعالجات بالوصول إلى الذاكرة المشتركة بنفس الوقت وذلك إذا كانت مواقع الذاكرة المحددة مـــن أجـــل القراءة منها أو الكتابة فيها مختلفة.

من ناحية أخرى يمكن أن نقسم فئة الحواسيب هذه إلى أربع فئات فرعية وفقاً لقدرة اثنـــين أو أكثـــر مـــن المعالجات على الوصول إلى موقع الذاكرة نفسه وبنفس الوقت وذلك كما يلي:

Exclusive-Read, Exclusive-Write (EREW) SM SIMD Computers (1-1-3)

في هذا النوع يكون الوصول إلى مواقع الذاكرة محدوداً بمعنى أنه لا يُسمح لمعالجين القراءة من الموقع نفسه مـــن الذاكرة أو الكتابة في الموقع نفسه من الذاكرة وذلك في الوقت نفسه.

Concurrent-Read, Exclusive-Write (CREW) SM SIMD (2-1-3) Computers.

يسمح هذا النوع من الحواسيب بالقراءة المتعددة للمعالجات من نفس موقع الذاكرة ولكن من أجل الكتابة فإنـــه يبقى محدوداً أي أنه لا يسمح لمعالجين بالكتابة في نفس موقع الذاكرة بالوقت نفسه.

Exclusive-Read, Concurrent-Write (ERCW) SM SIMD (3-1-3) Computers.

يسمح هذا النوع من الحواسيب بالكتابة المتعددة للمعالجات في نفس موقع الذاكرة ولكن الوصول مـــن أجـــل القراءة يبقى محدوداً إذ أنه لا يمكن لمعالجين القراءة من نفس الموقع من الذاكرة في الوقت نفسه.

Concurrent-Read, Concurrent-Write (CRCW) SM SIMD (4-1-3) Computers.

في هذا النوع من الحواسيب تتحقق ميزة القراءة المتعددة المتداخلـــة والكتابـــة المتعـــددة للمعالجـــات المتعددة.

إن خاصية القراءة المتعددة والوصول إلى نفس الموقع من الذاكرة لن يؤدي إلى أية مشاكل حيث أن كل من المعالجات التي تقرأ من ذلك الموقع تصنع نسخة من محتويات الموقع وتخزنها في ذاكرتها المحلية الخاصة ولكن المصاعب تظهر عند الوصول للكتابة المتعددة . فإذا حاولت عدة معالجات أن تخزن بيانات مختلفة في عنوان محدد بنفس الوقت فأي منها سينجح؟ وبمعنى آخر حتماً سيكون هناك طريقة لتحديد محتويات ذلك العنوان بعد عملية الكتابة.

مثال 1-2:

ليكن لدينا ملف كبير جداً يتألف من n إدخال منفصل وسنفترض للسهولة أن الملف غير مرتب بأي طريقة. والمطلوب تحديد فيما إذا كان العنصر x موجود في الملف أم لا من أجل إنجاز عملية قاعدة بيانات قياسية مثل القراءة والتعديل والحذف.

في حواسيب SISD التقليدية مثلاً تكون عملية استعادة x مطلوبة n مرة في أسوأ الحالات حيث يوجد في كل خطوة مقارنة بين x وإدخال الملف. وتكون الحالة الأسوأ عندما يكون x مساوياً لآخر إدخال أو غير مساوي لأي إدخال. ويصبح العمل أقل جودة بشكل متوسط كما يلي: إذا كانت إدخالات الملف موزعة بانتظام على المجال المعطى عندها تُطلَب نصف الخطوات العديدة لاستعادة x وهو العمل الذي نستطيع القيام به بشكل أسرع على حواسيب EREW SM SIMD مع N معالج حيث: $N<n$.

لنرمز للمعالجات بـ: $p_1,p_2,..,p_n$

نحتاج من أجل جعل جميع المعالجات تعرف قيمة x في البداية إلى استعمال تعليمة النشر أو الإعلان BROUDCASTING كما يلي:

1- p_1 يقرأ x ويبلغ p_2
2- بنفس الوقت p_1,p_2 تبلغ p_3,p_4 على التوالي
3- بنفس الوقت p_1,p_2,p_3,p_4 تبلغ p_5,p_6,p_7,p_8 على التوالي

وهكذا..

تستمر هذه العملية حتى تحصل جميع المعالجات على x حيث يتضاعف عدد المعالجات في كل مرحلة. يتطلب إعلان x إلى log n خطوة.

ليكن الملف الذي نبحث فيه عن x مقسم إلى ملفات جزئية يتم فيها البحث من قبل المعالجات بنفس الوقت: p_1 يبحث في العنصر n/N الأول وهكذا..

بما أن جميع الملفات الجزئية لها نفس الحجم فإننا سنحتاج إلى n/N خطوة في أسوأ الأحوال للإجابة عن استفسار x وبالنتيجة ومن أجل ذلك تتطلب الخوارزمية المتوازية log N + n/N خطوة في أسوأ الأحوال. لنأخذ موقع f يملك قيمة منطقية يحدد جانباً في الذاكرة المشتركة ويشير إلى أن أحد المعالجات قد وجد العنصر المطلوب وبالتالي فإن جميع المعالجات الأخرى تنهي عملها. مبدئياً تأخذ f القيمة false وعندما يجد المعالج x في ملفه الجزئي يضع في f قيمة true.

في كل خطوة من البحث تفحص جميع المعالجات قيمة f فإذا كانت true تتوقف.

نلاحظ أن هذا التعديل لم يأت بالسهولة فنحن بحاجة إلى log N خطوة للإعلان عن قيمة f في كل مرة تحتاجها المعالجات. أي إلى log N + n/N log N خطوة في أسوأ الأحوال. ولكي نستغل هذه النتيجة بدون زيادة زمن التنفيذ فإننا بحاجة إلى استعمال نماذج أكثر قوة تدعى حواسيب CREW SM SIMD. بما أن عمليات القراءة المتعددة مسموحة فإنها تأخذ خطوة واحدة لجميع المعالجات لكي تحصل على x وخطوة واحدة لتقرأ f في كل مرة تحتاجها. وهذا يقودنا إلى أن الحالة الأسوأ تأخذ n/N خطوة.

ربما يمثل الملف قاعدة بيانات نصية مع مئات الآلاف من البنود والفقرات كل منها يشمل عدة آلاف من الكلمات، وربما يكون ضرورياً البحث في مثل هذا الملف عن كلمة محددة x وهنا فإننا قد نجد أكثر من إدخال واحد مساوي لقيمة x ومن ثم أكثر من معالج يحتاج إبلاغ النجاح في نفس الوقت وهذا يعني أن اثنين أو أكثر من المعالجات ستحاول الكتابة في الموقع f في الوقت نفسه وهذا الشيء ممكن فقط في حاسبCRCW SM SIMD.

(2-3) حواسيب SIMD شبكة اتصالات:

يمكننا الحصول على نموذج مرن أكثر قوة من نموذج الذاكرة المشتركة يكون فيه كل زوج من المعالجات متصل بخط ثنائي الاتجاه ، حيث يمكن لعدة أزوج أن تتصل بآن واحد (ولكن بشرط ألا يحاول أكثر من معالج إرسال البيانات إلى معالج آخر أو تلقي البيانات من معالج آخر).
وبالتالي فإنه من المحتمل أن تستطيع جميع المعالجات أن تكون مشغولة بالاتصال جميع الوقت وهذا غير ممكن في ذاكرة مشتركة مقسمة إلى مقاطع R-block shared memory عندما يكون عدد المعالجات N أكبر من عدد المقاطع R .

(3-2-1) خصائص حواسيب SIMD شبكة اتصالات:

1- الكلفة (السعر): إذا كان السؤال المطروح : ما هو السعر المدفوع من أجل شبكة اتصالات تامة بـــ N معالج ؟ هناك (N-1) خط مسموح لكل معالج ومنه لدينا بشكل كامل N(N-1)/2 خــط. من الواضح أن مثل هذه الشبكة يكون سعرها غالي جداً وبشكل خاص من أجل قيمة كبيرة لــــ N. وهذا صحيح على الأخص إذا لاحظنا أنه مع N معالج فإن أفضل ما يمكن من أجل N ملف تخفيض عدد الخطوات المطلوبة في الخوارزمية التسلسلية.

2- الوثوقية Feasibility:

حتى لو استطعنا تحمل مثل هذا السعر العالي فإن النموذج غير واقعي عملياً مرة ثانية من أجل قيمـــة كبيرة لـــ N. بالإضافة إلى أنه هناك حد لعدد الخطوط التي يمكن وصلها للمعالج وهذا الحد يفرضـــه الحجم الفيزيائي الحقيقي للمعالج نفسه.

3- العلاقة بالنموذج SM SIMD:

أخيراً من الملاحظ أن نموذج الاتصالات التام[1] أضعف من حاسب بذاكرة مشتركة من أجـــل نفـــس السبب . مثلاً في الذاكرة المشتركة المقسمة إلى R مقطع لا يستطيع أكثر من معالج واحد الوصـــول المحدد بأن واحد إلى مقطع الذاكرة بالمساعدة مع معالج آخر. وهذا يؤدي إلى كلفة مماثلة تقريباً لكلفة حاسب SM SIMD (ولكن دون أن نحسب الكلفة من الدرجة الثانية لطرق ثنائية الاتجاه). وهذا يحبط غرضنا الأصلي في الحصول على آلة عملية أكثر.

(3-2-2) الشبكات البسيطة لحواسيب SIMD:

من حسن الحظ أنه في معظم التطبيقات هناك مجموعة جزئيـــة صـــغيرة مـــن جميـــع الارتباطـــات (الاتصالات) المزدوجة تكفي غالباً للحصول على أداء جيد.

سنلخص بإيجاز الشبكات الأكثر شيوعاً منها فيما يلي:

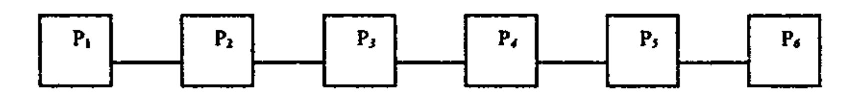
لنفترض أن لدينا معالجين يستطيعان الاتصال بعدد ثابت من الخطوات في حاسب SM SIMD . إن أي حاسب SIMD بشبكة اتصالات يمكن أن يُحاكى على نموذج سابق بدون أن يحتاج خطوات أكثر من الخطوات المطلوبة لإيجادها كما في السابق .

---

[1] نموذج الاتصالات التام: تتوزع في هذا النموذج مواقع الذاكرة المشتركة الـــ M بين N معالجاً ، كل منهـــا يتلقـــى M/N موقعاً. إضافة إلى أن كل زوج من المعالجات تتصل فيما بينها بخط ثنائي الاتجاه.

1- مصفوفة خطية Linear Array:

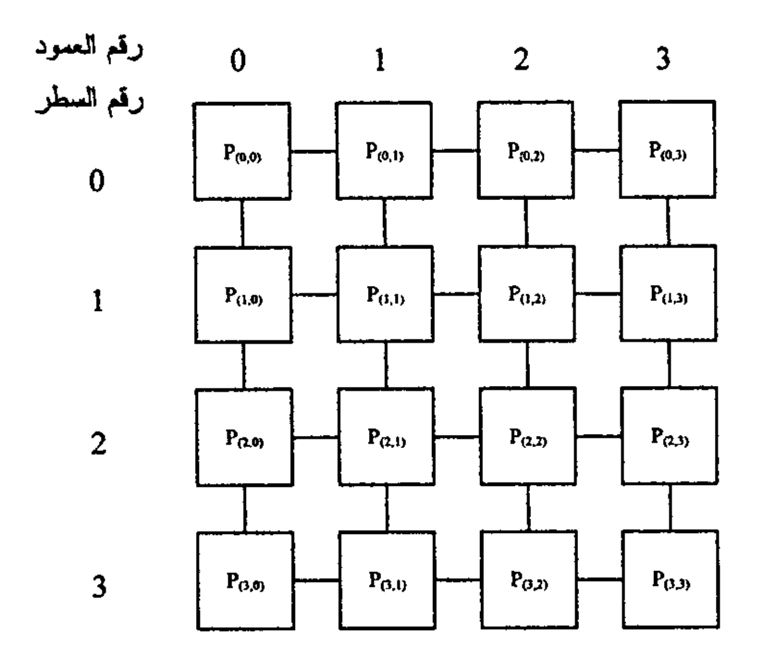
يمكن تمثيل وصل N معالج بشكل مصفوفة وحيدة البعد كما يظهر في الشكل (1-1) من أجل N=6.
هنا المعالج P_i مرتبط بمجاوريه P_{i-1} و P_{i+1} من خلال خط اتصال ثنائي الاتجاه.
كل من معالجات النهاية والمسماة P_1 و P_N تملك فقط مجاور واحد.



شكل (1-1) شبكة خطية من أجل N=6

2- مصفوفة ثنائية البعد Tow-Dimensional Array:

نحصل على شبكة ثنائية البعد بترتيب الـ N معالج في مصفوفة m×m حيث $m = N^{1/2}$ كما يظهر الشكل(1-2) من أجل m=4: المعالج في السطر j والعمود k يرمز بـ $p_{(j,k)}$ حيث: $0 \le k \le m-1$ و $0 \le j \le m-1$. هناك خط ثنائي الاتجاه يربط $p_{(j,k)}$ بمجاوريه $p_{(j-1,k)}$ و $p_{(j+1,k)}$ و $p_{(j,k+1)}$ $p_{(j,k-1)}$.
تملك المعالجات في الحدود الفاصلة للأسطر و الأعمدة عدد اقل من أربع مجاورات وبالتالي اتصالات أقل.
تعرف هذه الشبكة أيضاً بـ the mesh.
نلاحظ أنه في كل من المصفوفتين أحادية البعد وثنائية البعد تملك الشبكة خاصية مهمة: وهي أن جميع الخطوط في الشبكة تملك نفس الطول وهي غير موجودة في الاتصالات الأخرى التي سندرسها.



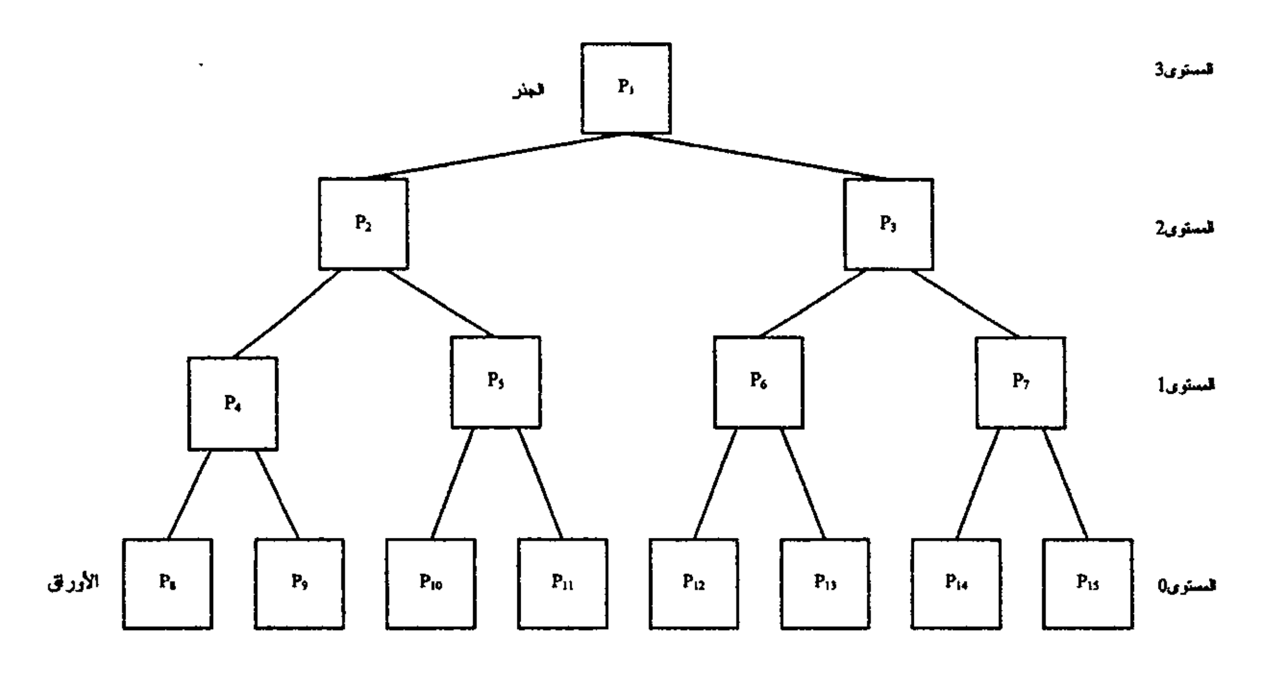
شكل1-2 شبكة اتصالات بمصفوفة ثنائية البعد (شبكية)

من أجل m=4

3- شجرة الاتصال Tree Connection

تكون المعالجات في هذه الشبكة بشكل شجرة ثنائية تامة ومثل هذه الشجرة تملك d مستوى مرقمة من 0 إلى d-1 و $N=2^d-1$ وينتهي كل من تلك بمعالج كما يظهر الشكل (1-3) من أجل d=4.

كل معالج في المستوى i يتصل بخط ثنائي الاتجاه بسلفه في المستوى i+1 ويتصل بخلفه في المستوى i-1 . المعالج الجذر في المستوى d-1 لا يملك سلفاً وكل من الأوراق في المستوى(0) لا يملك خلفاً أو أوراقاً.



شكل (1-3) شبكة اتصالات شجرية من أجل d=4

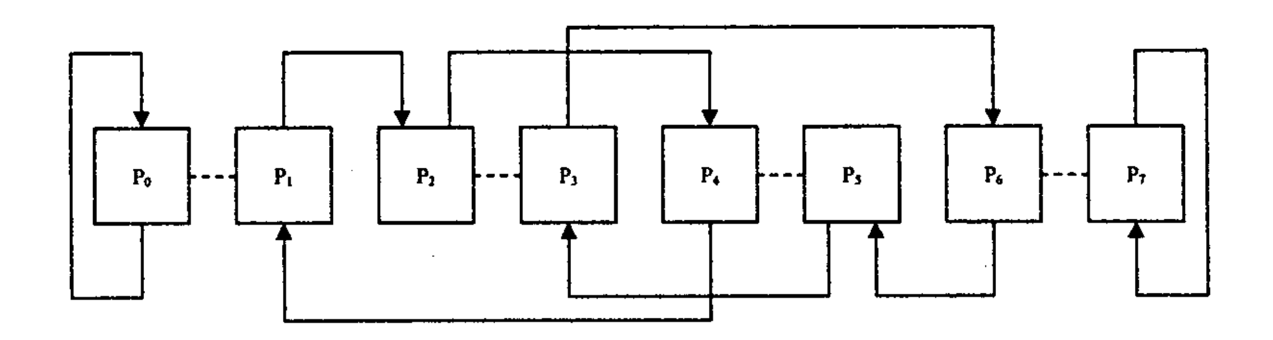
4- الاتصالات المختلطة التامة Perfect shuffle connection:

ليكن لدينا N معالج $P_0,P_1,P_2,..,P_{N-1}$ متوفرة حيث N هي قوة لــ 2 :$(N=2^x)$

يوجد في الاتصالات المختلطة التامة خط وحيد الاتجاه يربط P_i بــ P_j حيث:

$$J=\begin{cases} 2i & \text{من أجل} \quad 0 <= I <= N/2 - 1 , \\ 2i+1-N & \text{من أجل} \quad N/2 <= I <= N-1 , \end{cases}$$

كما يُظهر الشكل(1-4) من أجل N=8



شكل (1-4) شبكة اتصالات مختلطة تامة من أجل N=8

بشكل مكافئ نحصل على التمثيل الثنائي لـ j بتغيير دوري لـ i موضع واحد إلى اليسار ويوضح ذلك وفق ما يلي في الجدول 1-1.

المعالج p_i	P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
المعالج p_j	P_0	P_2	P_4	P_6	P_1	P_3	P_5	P_7

جدول 1-1 يبين الاتصالات لشبكة مختلطة تامة من أجل N=8

بالإضافة إلى أنه في بعض الأحيان تضاف إلى الشبكة خطوط ثنائية الاتجاه تصل كل معالج إلى الذي خلفه بشكل دوري على شكل ثنائيات في المثال:

(0,1)،(2,3)،(4,5)،(6,7) ثم (1,2)،(3,4)،(5,6)

وتدعى هذه الارتباطات بالارتباطات المتغيرة، وهي تظهر كخطوط مقطعة في الشكل(1-4). وفي هذه الحالة تعرف الشبكة بالارتباطات المتغيرة المختلطة.

5- الاتصالات التكعيبية Cube Connection:

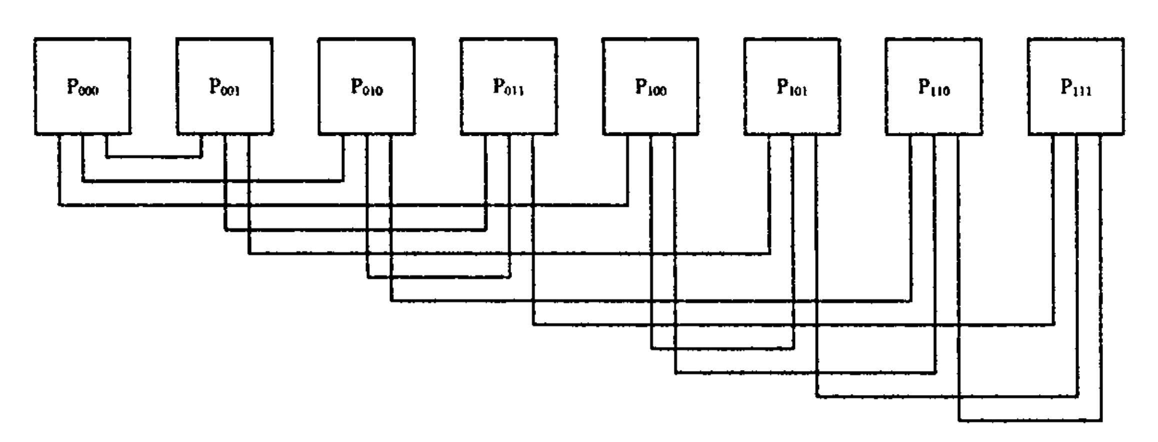
لنفترض أن $N=2^q$ من أجل بعض $q\geq1$ ولتكن الـ N معالج: $P_0,P_1,P_2,..,P_{N-1}$ مكعب من q بعد أو (hypercube) نحصل عليه بوصل كل معالج إلى q مجاور.

الـ q مجاور p_j من p_i معرفة كما يلي: نحصل على التمثيل الثنائي لـ j بتكملة البت المفرد في i. وهذا موضح في الجدول (1-2) لقيم i و j دليلي المعالجين المتجاورين حيث q=3 والفهرس لـ $p_0,p_1,..,p_7$ معطى في مجموعة رموز ثنائية.

i	j	j	j
000	001	010	100
001	000	011	101
010	000	011	110
011	010	001	111
100	101	110	000
101	100	111	001
110	111	101	011
111	110	101	011

جدول 1-2 يوضح التمثيل الثنائي لقيم أدلة المعالجات المتجاورة في شبكة اتصالات تكعيبية حيث q=3 و N=8

ونرى شكل الشبكة فيما يلي من أجل q=3 والفهرس لـ $p_0,p_1,..,p_7$ موضحاً بشكل رموز ثنائية.



شكل 1-5 شبكة اتصالات تكعيبية من أجل N=8

4- حواسيب MIMD (سلسلة تعليمات متعددة – سلسلة بيانات متعددة):

Multiple Instruction stream, Multiple Data stream (MIMD)

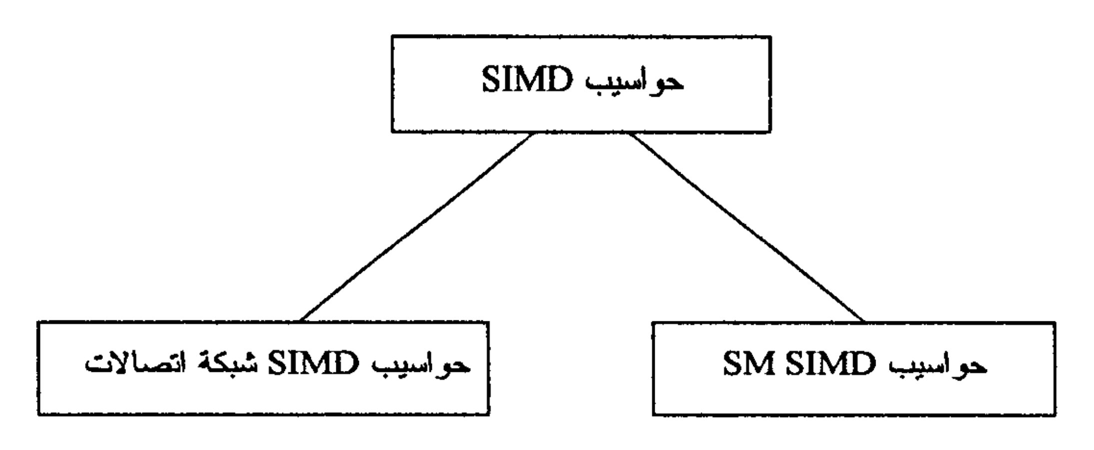
تعتبر هذه الفئة من الحواسيب الأكثر عموماً والأكثر قوة في مخطط الحساب المتوازي الذي يحدد فيما إذا كان التعدد في سلاسل التعليمات و/أو سلاسل البيانات، وفي هذه الحالة لدينا N معالج و N سلسلة تعليمات و N سلسلة بيانات. يكون لكل معالج وحدة تحكم خاصة به بالإضافة إلى ذاكرته المحلية الخاصة ووحدة منطقية وحسابية. وهذا يجعل هذه المعالجات أكثر قوة من المعالجات المستعملة في

حواسيب SIMD. يعمل كل معالج تحت تحكم سلسلة تعليمات موجهة إليه من وحدة تحكمه لذلك تنفذ المعالجات جميعها برامج مختلفة على بيانات مختلفة وتحل مسائل جزئية مختلفة من مسألة وحيدة وهذا يعني أن المعالجات تعمل بشكل متزامن.

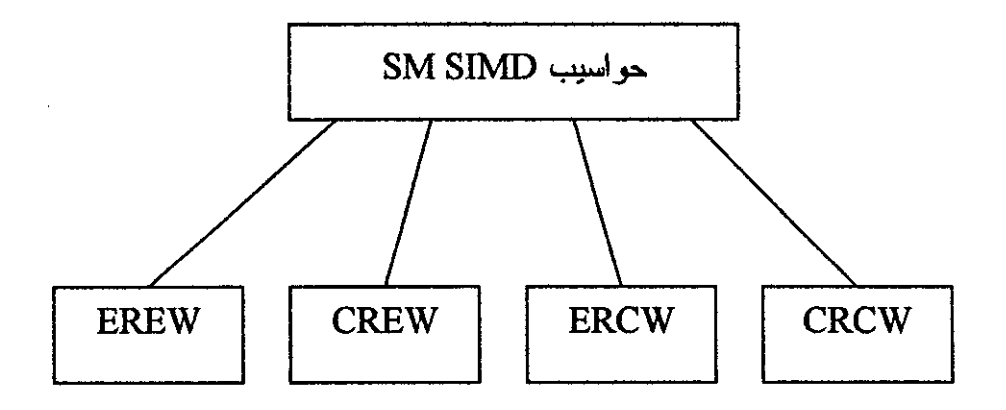
تنجز الاتصالات بين المعالجات كما في حواسيب SIMD من خلال الذاكرة المشتركة أو شبكة الاتصالات. تعرف حواسيب MIMD التي تتشارك في الذاكرة المشتركة بـ multiprocessor أو (tightly coupled machines) ويمكن تقسيمها حسب إمكانية القراءة المتعددة من الذاكرة أو الكتابة المتعددة في الذاكرة إلى الأصناف التالية من الحواسيب: EREW, CREW, ERCW, CRCW SM MIMD ؛ كما تعرف الحواسيب التي تتشارك من خلال شبكة الاتصالات بـ multicomputers أو (loosely coupled machines) وتشير هذه في بعض الأحيان إلى الأنظمة الموزعة.[SeGA-1989]

2-2-1 وصف الأجهزة المتوازية:

كما رأينا أن الأجهزة المتوازية تقسم إلى فئتين هامتين تبعاً لطريقة الاتصال بين وحدات المعالجة وذلك إما عن طريق الذاكرة المشتركة المقطعة إلى مقاطع block، أو عن طريق شبكة الاتصالات.
وتجدر الإشارة هنا إلى أن بعضاً من الفئات الجزئية التي سبق ذكرها يتصف بتوازٍ عام أكثر دقة على مستوى التعليمات الأولية نفسها مثل جهاز يدعى Data Flow حيث تنفذ كافة التعليمات على التوازي بعد أن يتم حساب معطياتها.
ويمكن تمثيل التصنيف السابق بالمخطط الآتي الذي يمثل تصنيف حواسيب SIMD وبشكل مشابه يكون تصنيف حواسيب MIMD.



التصنيف حسب طريقة اتصال المعالجات



التصنيف حسب الوصول إلى الذاكرة من أجل القراءة و الكتابة

3-2-1 جهاز CRAY-XMP متعدد المعالجات:

يتكون جهاز CRAY-XMP من أربعة معالجات شعاعية خطية أنبوبية بذاكرة مشتركة ويستخدم التوازي العام MIMD والتوازي المحلي SIMD.[MB.H,1998]

1- البنية العامة لجهاز CRAY-XMP:

يقسم العمل المنجز على هذا الجهاز إلى مهمات تنفذها عدة معالجات في آن معاً ويعمل كل معالج على المهمة المخصصة له بشكل تتابعي كما يستطيع الوصول للذاكرة الرئيسية عبر المسارات المخصصة له ، حيث يخصص لكل معالج مسارين للقراءة ومسار للكتابة وآخر للإدخال والإخراج (عن طريق معالج خاص) ولا توجد أية حماية أو مزامنة ضمنية للذاكرة المركزية. كما يحتوي على معالج داخلي مهمته إدارة الاتصالات بين المعالجات. ويجب أن نأخذ بعين الاعتبار في مستوى البرمجة صراع المعالجات للوصول إلى الذاكرة.

لنأخذ الآن بعض الصفات الرقمية لجهاز CRAY-XMP2:

1. تستطيع المعالجات إنجاز عمليات شعاعية وسلمية.
2. تمتاز الذاكرة المشتركة بأنها من حجم كبير وتبلغ استطاعتها(456 435 268) كلمة ، طول كل كلمة bits 64 مقسمة إلى كتل مستقلة ولكل كتلة مدخلها الخاص بغية زيادة سرعة التدفق للذاكرة.

3. تبلغ مدة الدورة الأساسية في جهاز CRAY-XMP2 4.1 ns وتنتج العمليات الشعاعية نتيجة في كل دورة في حين أنه يجب علينا من أجل الحسابات الأخرى أن نأخذ بعين الاعتبار دورة فعلية معادلة لدورتين أساسيتين.
4. يؤمن المحيط البرمجي لجهـاز CRAY-XMP2 بواسـطة متـرجمين: المتـرجم CFT2 والمترجم الجديد CFT77، كما أن نظام الاستثمار المعتمـد هـو نظـام UNIX SYSTE .VATT

2- جهاز CRAY-XMP كآلة شعاعية:

يمتلك جهاز CRAY-XMP بصفته جهازاً شعاعياً تعليمات شعاعية تكون خطية أنبوبية : أي تقسم التعليمات لمهمات عنصرية ينفذ كل منها عامل متخصص. ويمكن مقارنة الخط الأنبوبي بخط تجميع السلع حيث تمر السلعة على عدة مراكز عمل وعندما يزود الخط بالمواد الضرورية فإن السلع الجاهزة تخرج من الخط في كل دقيقة بالرغم من أن تصنيعها يحتاج لعدة ساعات. تستخدم التعليمات الشعاعية سجلات شعاعية يتكون كل منها من 64 كلمة ، طول كل كلمة 64 خانة.

ومما تجدر الإشارة إليه أن أول أجهزة CRAY قد ظهرت في عام 1985 وكان الجهاز مكوناً مـن معالجين. ومنذ ذلك التاريخ تم تطوير هذه الأجهزة. ففي السادس والعشرين من آذار 1997 ابتدأ فـي مركز الحساب العلمي التابع لمفوضية الطاقة الذرية في غرونوبل باستخدام الحاسوب الجديد الموازي SGI-CRAYT3E الذي يتميز بسعته الكبيرة. لهذا الحاسوب 256 معالجاً ويعمل بتردد 375 ميغـا هرتز ولكل معالج منها ذاكرته الخاصة، ويستطيع العمل بمفرده إلا أن شبكة ربط عالية الأداء تتـيح لها الاتصال فيما بينها، وهكذا يمكن لباحثي مختلف الإدارات العملياتية في هيئة الطاقة الذرية الاستفادة من قدرة حسابية تعادل عملياً عشرة أضعاف من قدرة النموذج السابق الذي جُهز به المركز ذاته.

يستطيع هذا الحاسوب العملاق الذي يصل حجم ذاكرته إلى 32 مليار بايت أن ينجز في الواقع حتـى 190 مليار عملية بالثانية الواحدة وبفضله وهو الأكبر استطاعة من أمثاله من هذا النوع المستخدم في فرنسا حتى هذا اليوم سيكون ممكناً ومن الآن فصاعداً المباشرة بعملية محاكاة تتطلب حسابات ضخمة مع التأكيد على حصول تصحيح سريع للنتائج بتجنيد جميع المعالجات أو بعض منها لمعالجة قضية أو مسألة واحدة.

4-2-1 نمذجة الأنظمة متعددة المعالجات:

[HELR,1998], [KBag,1997]تعتبر نمذجة الأنظمة ذات دور هام في تحديد فعالية الأنظمة وتعقيدها وتعطي رؤية معمقة عن سلوكية الأنظمة. ويمكن أن نعتمد أسلوباً لفهم مشــكلات التــزامن والتوازي وهو المحاكاة التي تعتبر طريقة عامة تقريباً على كل الأنظمة إذ أن معظم سلوكيات الأنظمة تتشابه فيما بينها وبالرغم من أن المحاكاة اقتصادية أكثر من إنشاء الأنظمة مباشرة إلا أن هذا الأسلوب يعتبر بطيئاً وذو كلفة مرتفعة نسبياً إضافة لذلك فهي تعطي معلومات محدودة عن العلاقة المنطقية بين سلوكية النظام المحاكى وصفات النظام الحقيقي.

1- نماذج الحساب المتوازي:

يمكن أن نوضح مفهوم الحساب المتوازي بتعريف بعض النماذج سنقوم باستعمالها لتقديم أفكار ونتائج تشكل أساسات نظرية الحساب المتوازي.

2- نظريات الحساب المتوازي:

[MJQu,1994] وهي تعرّف أن المكان والزمان هما المصدران الأساسيان الأكثر أهمية المعرفان لتعقيد الحساب التسلسلي. وفي الحساب المتوازي تبقى أهمية الزمن نفسها: حيث أن السبب الرئيســي لدراسة الحساب المتوازي هو للحصول على زيادة سرعة حل المسائل الحسابية speed up ، وبشكل بديهي يمكننا أن ندرك ببساطة أنه إذا توفر لدينا وحدات معالجة أكثر فإن الزمن المطلوب لحل مسألة معطاة ينخفض، وهذه الحقيقة تجعل وحدات المعالجة هي المتطلب الرئيســي الثــاني فــي الحســاب المتوازي.

بالإشارة إلى النموذج الذي نعالجه فإن هناك العديد من مجموعات وحدات المعالجة لكمية متوفرة مــن المكونات الصلبة. وهذه المكونات الصلبة للحاسوب هي الوحدة الأكثر عمومية في القياس والتي يمكن في بعض الحالات أن تتطابق مع عدد وحدات المعالجة.

الفكرة القائلة أن الزمن والمكونات الصلبة هما القياسان الرئيسيان للحساب المتوازي مستندة على عدة نتائج مرتبطة بمصادر تسلسلية للمكان والزمان على التوالي مع الزمن والمكونات الصلبة في الحالــة المتوازية.

وهذا التطابق الموجود بين المكان والزمان في أي نموذج اعتيادي للحساب المتوازي يعطـــي نظريـــة الحساب المتوازي، وبالأخذ بعين الاعتبار أن جميع نماذج الحساب التسلسلي تتصل ببعضـــها بشـــكل متعدي يمكننا عندئذٍ أن نعبر عن فرضية الحساب المتوازي كما يلي:

إن أي آلة حساب تورنك مع كلفة مكان S(n) يمكن أن تُحاكى على نموذج عادي للحساب المتـــوازي بكلفة زمنية $T(n)=O(S(n)^c)$ ، وبالعكس فإن أي حساب على نموذج عادي للحساب المتوازي مـــع كلفة زمنية T'(n) يمكن أن يُحاكى بواسطة آلة حساب تورنك بكلفة مكانية $S'(n)=O(T'(n)^d)$ حيـــث d,c هي ثوابت تابعة للنموذج.

3- الحسابات المتوازية والنماذج اللوغاريتمية:

تعتمد معظم لغات البرمجة عالية المستوى على آلة خاصة صممت في بنية الحاسب المعتاد. تدعى هذه اللغات قاعدية imperative ، ويعرف البرنامج على أنه أوامر تسلسلية أساسية للمعالج، كما أن بنيـــة الحاسب المعتاد نفسها تعتمد على النظرة التسلسلية للحسابات. ومن وجهة منطقية يمكن أن نعتبر بعض خطوط البرنامج مستقلة حتى إذا كانت تسلسلية فيزيائياً.

لنأخذ برنامج ضرب مصفوفتين من الحجم n وهو برنامج خطي مستقيم يعبر عن الحالة n=2 وهـــو كالتالي:

```
For  i=1 until   n  do
        For  j=1 until   n  do
        Begin
                c(i,j)=0;
                For  k=1 until   n  do
                        c(i,j) = c(i,j) + a(i,k) * b(k,j)
        end;
```

سنبين الآن الخطوات التي يتم فيها تنفيذ البرنامج وكيف تأخذ عناصر المصفوفة الناتجة قيمها وذلـــك فيما يلي:

1   c(1,1) → 0;

2 $c(1,1) \rightarrow c(1,1) + a(1,1) * b(1,1);$

3 $c(1,1) \rightarrow c(1,1) + a(1,2) * b(2,1);$

4 $c(1,2) \rightarrow 0;$

5 $c(1,2) \rightarrow c(1,2) + a(1,1) * b(1,2);$

6 $c(1,2) \rightarrow c(1,2) + a(1,2) * b(2,2);$

7 $c(2,1) \rightarrow 0;$

8 $c(2,1) \rightarrow c(2,1) + a(2,1) * b(1,1);$

9 $c(2,1) \rightarrow c(2,1) + a(2,2) * b(1,2);$

10 $c(2,2) \rightarrow 0;$

11 $c(2,2) \rightarrow c(2,2) + a(2,1) * b(1,2);$

12 $c(2,2) \rightarrow c(2,2) + a(2,2) * b(2,2);$

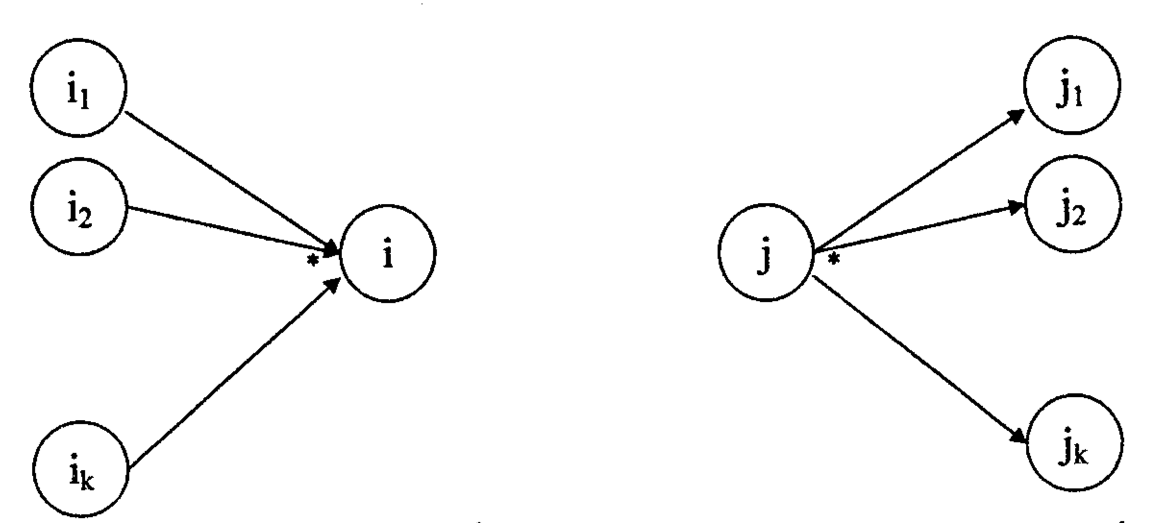
من الواضح في تحقق البرنامج أن بعض التعليمات يتعلق بالآخر ولذلك نشكل السلسلة من التعليمات الأكثر لزوماً.

نلاحظ أن البرنامج يمكن أن يقسم إلى أربع مقاطع مختلفة كل واحد منها يحسب n عنصر من المصفوفة الناتجة.

ومن الواضح أنه يمكننا حساب العنصر c(2,2) أولاً ومن ثم نحسب العنصر c(1,1). كما نجد أن هناك اختلافاً بين حالتين وهما حساب العناصر c(1,1) و c(2,2) وبالمثل c(1,2) و c(2,1) حيث يمكن حسابهما على التوازي وذلك على افتراض توفر المصادر الحسابية المناسبة. لنأخذ الآن الشكل التالي الذي يرمز إلى أن التعليمة i تسبق التعليمة j:

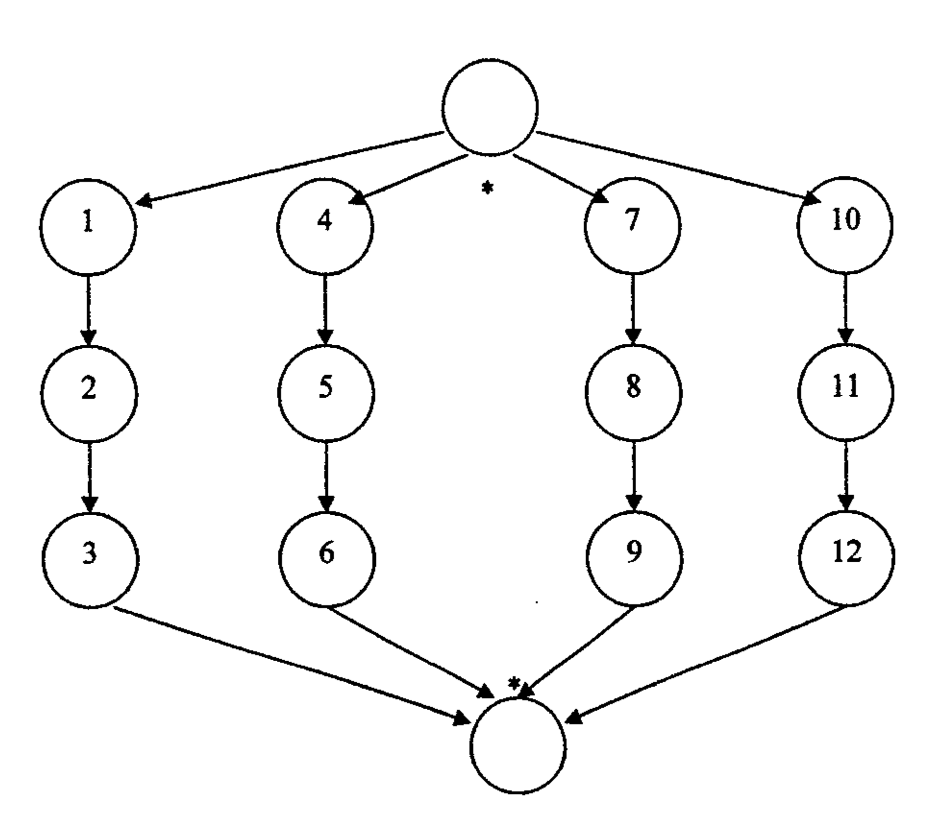


ولكي نرمز إلى أن التعليمات $i_1,...,i_k$ تسبق التعليمة i وبالعكس التعليمات $j_1,...,j_k$ مسبوقة بالتعليمة j نكتب على التوالي:

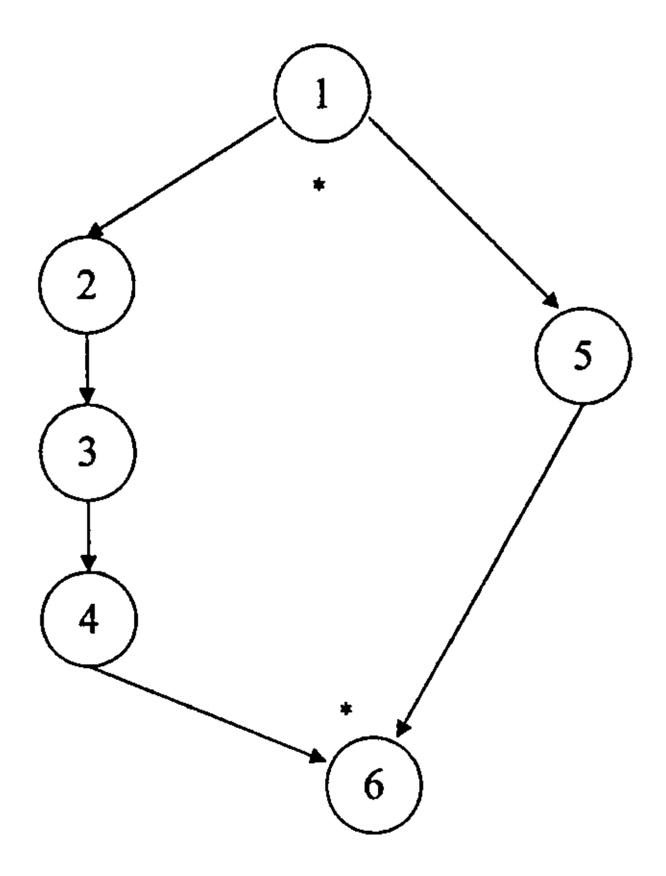


يمكننا أن نصيغ بهذه الرموز بشكل بياني الترتيب الذي يمكن أن تنفذ فيه تعليمات البرنامج ويدعى هذا الترميز ببيان الترتيب الجزئي partial ordering graph.

يبين الشكل التالي بيان الترتيب الجزئي لبرنامج ضرب مصفوفتين الذي أوردناه سابقاً:



الشكل (1-6) بيان الترتيب الجزئي لبرنامج ضرب مصفوفتين



شكل (1-7) بيان ترتيب جزئي

تكون التعليمتان مستقلتان إذا كانتا غير متصلتين بمسلك (طريق) على الغراف (البيان) وعندئذ يمكن تنفيذهما على التوازي.

من الواضح أنه عندما يسري برنامج على التوازي فإن النتائج يجب أن تتدفق متزامنة ، وهذا ضروري لكي نتأكد من أن الحسابات صحيحة.

لنفترض كمثال بيان الترتيب الجزئي في الشكل(1-7) ولنفترض أن كل تعليمة تحتاج نفس الزمن فإن المسلك (4,6) (3,4) (2,3) (1,2) سيكون أطول من المسلك (5,6) (1,5).

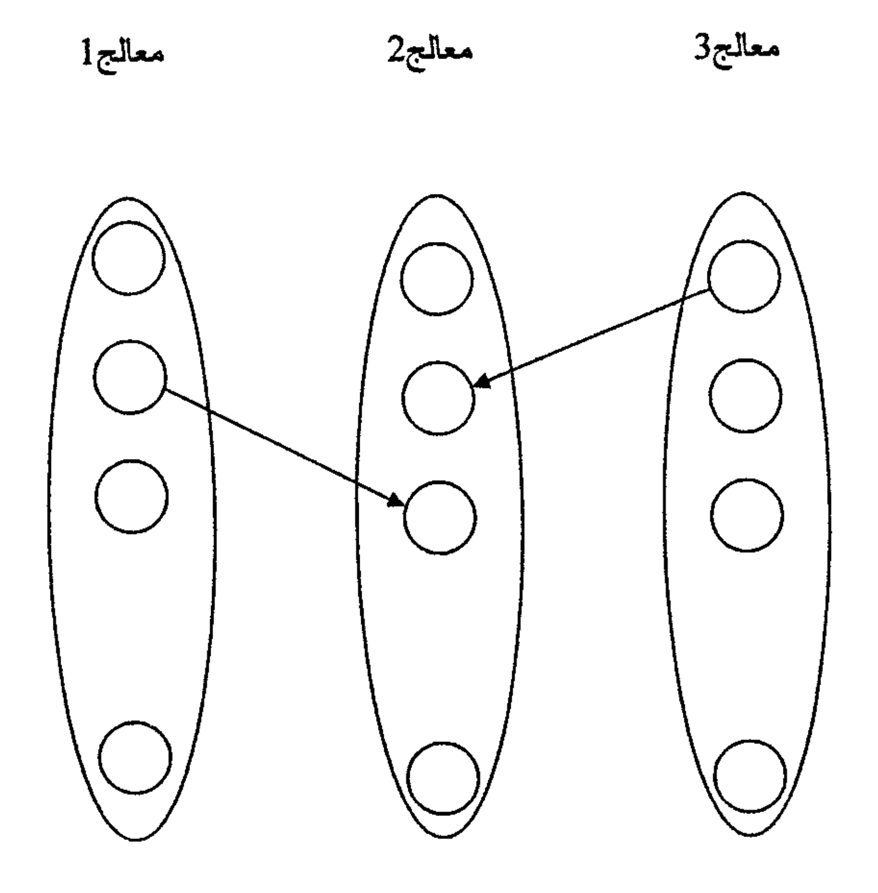
إذا كانت التعليمات على طول المسلكين تنفذان على التوازي فإنه يجب أن نزود البرنامج بتقنية تزامن ما بين تدفقي البرنامجين ليضمن أن التعليمة 6 لم تنفذ قبل التعليمتان 4 و 5 .

يمكن أن نجعل جميع المسالك بطول متساوي باستخدام حل وحيد وهو إضافة تعليمات خالية مثلاً التعليمة : $x \leftarrow x$ حيث x هو أي متغير ، ونرمز للتعليمة الخالية في بيان الترتيب الجزئي بتعليمة حلقية.

إن بيان الترتيب الجزئي لتعليمات برنامج يعرف حساباً متوازياً ممكناً، ومن الواضح أن قدرة تنفيذ التعليمات المستقلة يعتمد على توفر المصادر الحسابية. علاوة على ذلك يجب أن نعرف معياراً يحدد التعليمات للمعالجات، وبالاعتماد على نموذج الحساب المتوازي تحدد التعليمات تبعاً لأحد البندين التاليين:

1- كل عقدة بيان (مثلاً كل تعليمة) محددة لمعالج مختلف. وفي هذه الحالة فإن الأقواس الداخلة (الخارجة) من العقدة يمكن أن تظهر كاتصالات من (إلى) ذلك المعالج الذي يتلقى (ينقل) البيانات.

2- تحدد المسالك المختلفة (ليس بالضرورة مسالك مستقلة) لمعالجات مختلفة ، في هذه الحالة تنفذ التعليمات على نفس المسلك من قبل نفس المعالج وينشئ المعالج حساباً تسلسلياً. تحدث الاتصالات فيما بين المعالجات في تلك العقد حيث تعرّف المسالك المختلفة العلاقات فيما بينها كما في الشكل(1-8):



شكل (1-8) تحديد التعليمات للمعالجات

في الحقيقة نحن نبحث عن طريقة نظامية للحصول على بيان من بيان الترتيب الجزئـي للتعليمـات لبرنامج معطى، حيث يتم في هذا البيان زيادة الاحتمال المتوازي إلى الحـد الأعلـى مـن الحسـاب التسلسلي. ومن جهة أخرى نلاحظ أنه ليس بالإمكان الحصول دائماً على خوارزميات متوازية فعالـة من الخوارزميات التسلسلية الفعالة. إن مسألة موازاة الخوارزميات التسلسلية هي مسألة مهمة عنـدما ندرس المترجمات (المصنّفات compilers) أو الآلات الشعاعية (vector machines).

إن معظم لغات البرمجة تتماثل من وجهة النظر الأساسية للحسابات ويمكن أن تظهر تعليمة الحاسـب التقليدي كأمر للمعالج وكدالة ، متغيرات هذه الدالة هي موقع الذاكرة (أو القـيم المخزنـة فـي تلـك المواقع). إذا كان لدينا تعليمة I_j تستعمل قيم مواقع الذاكرة k_j وتحسب محتويات مواقـع الـذاكرة h_j عندئذ نكتب ذلك ترميزاً:

$$I_j \; : \; M^{kj} \; \rightarrow \; M^{hj}$$

حيث M مجموعة من مواقع الذاكرة تتطابق في النموذج RAM مع المسجلات ونكتب:

$$\text{Load} \; : \; M \; \rightarrow \; M$$

$$\text{Add} \; : \; M^2 \; \rightarrow \; M$$

ومن المعروف أن التعليمات المستقلة يمكن أن تنفذ على التوازي ولدينا:

لتكن D(I) المحتواة في M ترمز إلى مجموعة مواقع الذاكرة التي تستعمل قيمها في بعض التعليمات I.

ولتكن R(I) المحتواة في M ترمز إلى مجموعة مواقع الذاكرة التي عُدّلت محتوياتها من قبل التعليمة I.

إذا كانت I من النوع I_j فإن عدد العناصر في المجموعة D(I) و R(I) هو k_j و h_j على التوالي.

4- شروط برنشتاين:

لكي تكون التعليمتان I و J مستقلتين فإنه من الكافي أن تتحقق العلاقات التالية والتي تـدعى شـروط برنشتاين:

$$R(I) \cap R(J) = \Phi$$
$$D(I) \cap R(J) = \Phi$$
$$R(I) \cap D(J) = \Phi$$

عندما لا تتحقق شروط برنشتاين عندئذٍ يوجد تبعية منطقية بين I و J وللحصول على حساب أكيد يجب أن ينجز التسلسل.

سنحدد الآن طريقة للحصول على بيان الترتيب الجزئي الذي يزيد الحد الأعلى لاحتمال التوازي للحساب التسلسلي: نعمل على تطبيق شروط برنشتاين على كل زوج من التعليمات فإذا لم يكن أي منها غير محقق عندئذٍ يكون البيان يصف الحساب المتوازي بشكل أعظمي. كمثال نأخذ برنامج ضرب مصفوفتين من الحجم n المذكور سابقاً ونلاحظ أن استخدام برنامج مختلف يمكننا من الحصول على حساب موصوف بزمن متوازي متناسب مع log n أكثر من أن يتناسب مع n. وكما رأينا أن كل عقدة في البيان تشير إلى معالج مختلف وكل قوس يدل على بيانات متبادلة على طول بعض الاتصالات الفيزيائية.

إن عدد العقد يعطي مقياساً لمصادر المكونات الصلبة المطلوبة من قبل الحساب طالما أن طول المسلك الأطول يقيس زمن التوازي (parallel time).

5- تعريف النموذج اللوغاريتمي:

يبدأ هذا النموذج من شروط برنشتاين لاحتمال توازي الخوارزميات (ولا يهتم بالتفاصيل) ويمكن وصفه بالحقائق التالية:

1- يمكن أن يستعمل أي عدد من المعالجات في أي لحظة.
2- يمكن أن ينفذ كل معالج أي تعليمة (رياضية أو منطقية) في وحدة زمن.
3- لا يوجد كلفة من أجل الوصول للبيانات.
4- لا يوجد كلفة للاتصال بين المعالجات.

أما المصادر الأساسية المطلوبة من النموذج فهي عدد الخطوات المتوازية (عدد وحدات الزمن) وعدد المعالجات المستعملة كدالة في حجم المسألة.

مثال(1-3):

يمكن أن نتحقق من قواعد تعريف النموذج اللوغاريتمي ونطبقها على خوارزمية حساب ناتج جداء مصفوفتين وتحليل الكلفة الناتجة:

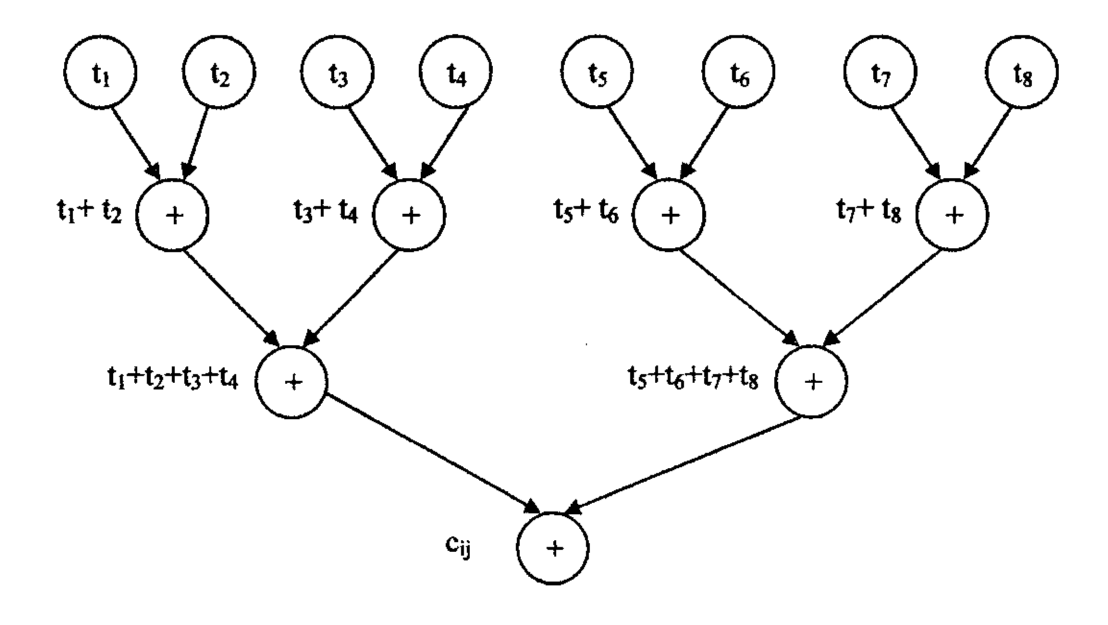
الدخل: مصفوفتين $A = (a_{ij})$ و $B = (b_{ij})$ من الحجم n.

الخرج: مصفوفة $C = (c_{ij})$ من الحجم n حيث: C=AB.

الخطوات:

1. حساب الـ n^3 ناتج على التوازي حيث: $t_{ikj} = a_{ik} * b_{kj}$
2. حساب الـ n^2 مجموع على التوازي حيث: $c_{ij} = t_{i1j} + t_{i2j} + \ldots + t_{inj}$ ؛ $i,j=1,\ldots,n$

نستعمل المخطط التالي للتوضيح ونرمز فيه بـ t_k بدلاً من t_{ikj} وفيه n=8 وهو في الشكل(1-9).



الشكل(1-9) حساب مجموع 8 أعداد

- **الكلفة:**

الخطوة 1 تتطلب وحدة زمن واحدة و n^3 معالج.

الخطوة 2 تتطلب $\lceil \log n \rceil + 1$ خطوة و n^3 معالج مطلوب.

عندما يكون العدد p من المعالجات ثابتاً يكون التوازي منتهياً.

يمكن تحليل إنجاز الخوارزمية عن طريق مقياسين آخرين وهما: زيادة السرعة والفعالية.

- زيادة السرعة speed up ونرمز لها $S_p(n)$ تعرف بأنها النسبة بين كلفة الزمن لأفضل خوارزمية تسلسلية متوفرة وكلفة الزمن لخوارزمية التوازي التي نحللها. عندما تكون خوارزمية التوازي تملك على الأغلب p معالج متوفر فإن: $S_p(n) = T_1(n) / T_p(n)$ حيث n هو حجم المسألة.

نلاحظ أن أفضل حالة لهذا الشرط هو عندما يكون $S_p(n) = p$ وذلك من الصعب الوصول إليه عملياً. والمطلوب هو الحصول على مسألة تَحُل بزمن تسلسلي $T(n)$ أن تُحَل بزمن متوازي $T(n)\backslash p$ حيث p هو عدد المعالجات المتوفرة. إضافة إلى أنه يوجد زمن ما لا محالة في الحساب المتوازي يستعمل لتنسيق فعالية المعالج. ويتحدد التوازي تبعاً لطبيعة المسألة نفسها.

- الفعالية (efficiency) ونرمز لها $E_p(n)$ وهي النسبة بين زيادة السرعة وعدد المعالجات: $E_p(n) = S_p(n)\backslash p$ من الواضح أن $E_p(n) \leq 1$ وتتحقق أفضل فعالية عندما تطبق إشارة التساوي.

يطبق مفهوم زيادة السرعة والفعالية عند دراسة التوازي وفي تحليل الخوارزميات عندما يزداد عدد المعالجات مع حجم المسألة، في هذه الحالة نجعل $p=p(n)$ وتحسب زيادة السرعة والفعالية كما لو أن p ثابت.

في هذا النموذج نحاول الحصول على المعلومات التالية:

1- القيم من الحجم n التي من أجلها يكون إنجاز الخوارزمية أفضل.
2- السلوك المقارب لزيادة السرعة والفعالية.

في المثال السابق تعطى زيادة السرعة والفعالية كما يلي:

$$S_n^3(n) = n^3 / (\lfloor \log n \rfloor + 1)$$

$$E_n^3(n) = 1 / (\lfloor \log n \rfloor + 1)$$

نلاحظ أن الفعالية تتلاشى عند النهاية $n \to \infty$ هذا يعني أن n هي الأعظمية الزمن الأطول الذي تكون خلاله معظم المعالجات غير فعالة.

وهكذا نجد أن الاقتراحات السابقة هي احتمالات لدراسة الخوارزميات المتوازية. أحدها للتفكير في العدد p لوحدات المعالجة المتوفرة كثابت ولمحاولة تأليف خوارزميات متوازية مع زمن تنفيذ $T_p(n)$ يرفع زيادة السرعة إلى الحد الأعلى. من جهة أخرى p يمكن أن يكون كدالة في حجم الإدخال أي أن $p = p(n)$ ويمكن تجريب خوارزميات تحتاج أقل احتمال لزمن متوازٍ.

6- الدارات المنطقية:

[JJAJ,1992] قبل أن تبدأ الدراسات حول التوازي كانت تستعمل الدارات المنطقية بشكل كبير عندما ندرس تعقيد الحسابات ونستعمل طول الدالات المنطقية (البوليانية). من ناحية ثانية فإننا نرى أن الحواسيب الاعتيادية هي آلات متوازية عندما ندقق أكثر بالتفاصيل ففي الحقيقة إن كل حاسب مصنوع من دارات الكترونية تحوي الملايين من العناصر المفتاحية وخطوط النقل كلها تعمل على التوازي ، والدارات المنطقية هي نموذج منطقي من الدارات الالكترونية. يعمل هذا النموذج على مسائل تمثل بسلسلة من الخانات ولهذا السبب فإن الدارات المنطقية تميز نفسها بأنها معالجات خطية (string processors). تحول جميع المعلومات الداخلة والخارجة من حاسب حقيقي إلى رموز في سلسلة من الخانات الثنائية، ومع بعض الترتيب في البنية تصبح دالة على بيانات ، بمعنى أن السلسلة تعتمد على التفسير (الترجمة) interpretation وهو خارجي بالنسبة للآلة. وبما أن تنفيذ العمليات يتم في مستوى الخانة فإنه عندما نعبر عن خوارزمية بدارة منطقية فإن كلفتها تدعى [كلفة الخانة bit cost] بينما يدعى تعقيدها [تعقيد الخانة bit complexity] وهي تعد مقياساً لكلفة التوازي. إن الخصائص السابقة تجعل نموذج الدارة المنطقية قاعدة لدراسة تعقيد الحسابات المتوازية.

7- تعاريف:

❖ تعريف 1:

لتكن لدينا دارة منطقية α ولها n دخلاً و m خرجاً هي بيان حلقي ومعنون وموجه $\alpha=(V,E)$ عناصر مجموعة العقد V مرقمة من 1 إلى |V| ومقسمة إلى أربع مجموعات منفصلة:

1- عقد الدخل input nodes
2- عقد ثابتة constant nodes
3- عقد عمليات operation nodes
4- عقد الخرج output nodes

-الـ n عقدة دخل لا تملك أقواساً داخلة إليها. كل واحدة منها معنونة برمز متغير مختلف. فيما يلي سنفترض أن هذه العقد مرقمة من 1 إلى n وأن عنون العقدة i هو x_i حيث: $(i=1,..,n)$.

-العقد الثابتة لا تملك أقواساً داخلة إليها وكل منها معنون بـ 0 أو 1. ويمكن أن تكون مجموعة العقد الثابتة خالية (فارغة)، وإن وجدت هذه العقد فإنها ترقم من n+1 إلى n+c حيث $c \geq 1$.

-الـــ m عقدة خرج لا تملك أقواساً خارجة منها وتملك تحديداً قوس دخل واحد. كـل منهـا معنـون بواسطة رمز متغير مختلف. سنفترض أن هذه العقد مرقمة من $|V| - m + 1$ إلى $|V|$ وأن عنوان العقدة i هو y_i حيث: $i = |V|-m+1,\ldots, |V|$.

عقد العمليات تملك الأقواس الداخلة والخارجة. كل منها معنون برمز لدالة منطقية مع واحد أو أكثـر من الوسطاء، ويعطى عدد الوسطاء الموجودة بواسطة عدد الأقواس الداخلة إلى العقدة. سنفترض فيما يلي أن عنوان العقدة i مع j_i قوس دخل هو $f_{l(i)}$ حيث:

$$f_{l(i)} : \{0,1\}^{j_i} \rightarrow \{0,1\}, \; i = n+c+1, \ldots, |V|-m$$

المجموعة $F=\{f_1,\ldots,f_r\}$ من جميع الدالات التي تعنون عقد العمليات في α تدعى قاعدة الدارة. وفي حالة خاصة يمكن أن تكون مجموعة عقد العمليات فارغة.

عدد الأقواس الداخلة (الخارجة) لعقدة معطاة يدعى fan-in (fan-out) لتلك العقدة . ويعتبر أعظـم fan-in (maximum) لجميع العقد هو fan-in للدارة، ويحدد عن طريق الدوال القاعدية.

سنبين ذلك على قاعدة تدعى القاعدة القانونية وهي مكونة من عمليات الجمـع المنطقـي والضـرب المنطقي والإتمام المنطقي والتي نرمز لها بالرموز التالية على التوالي:
($\vee$ و $\wedge$ و $\neg$).

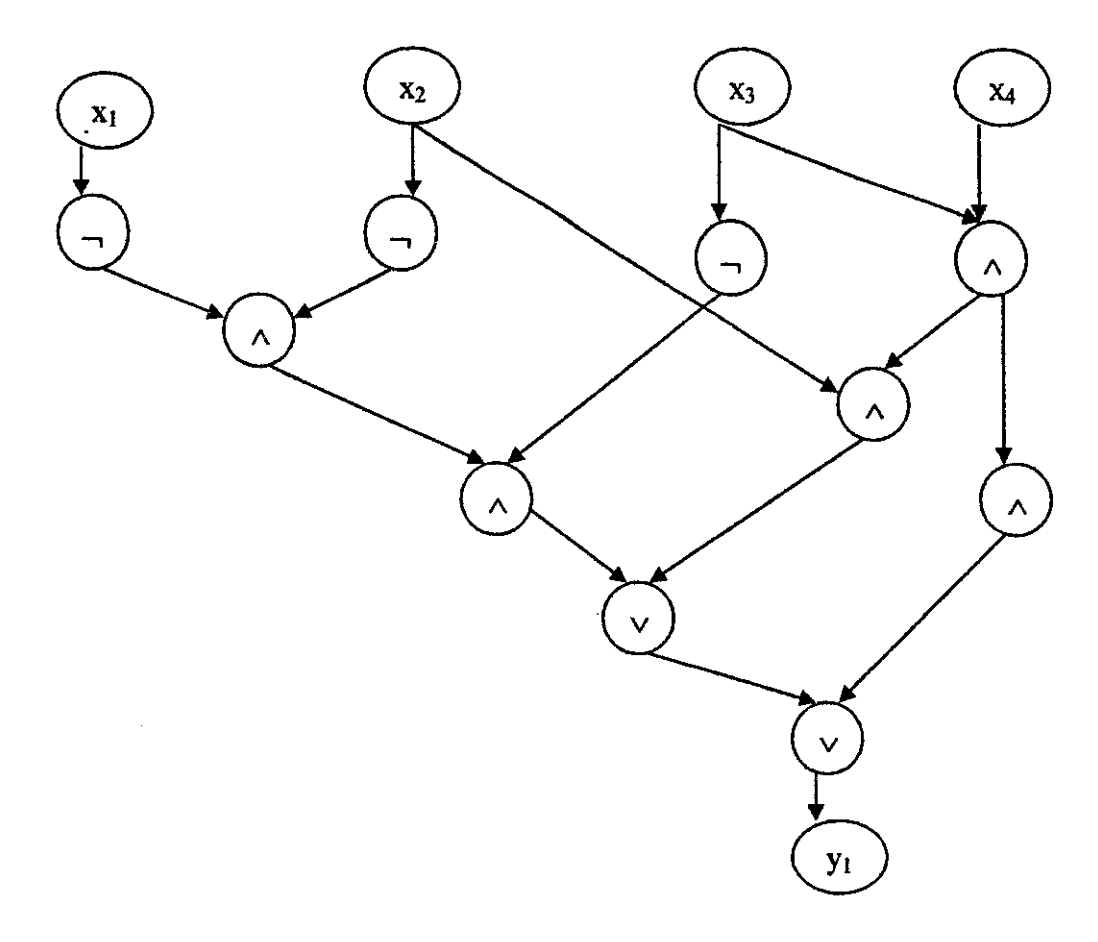
من المعروف أن القاعدة القانونية هي قاعدة تامة ، على سبيل المثال إن أي دالة منطقية يمكن أن نعبر عنها بواسطة عمليات الجمع المنطقي والضرب المنطقي والإتمام المنطقي فقط. الـــ fan in للدارات المعرفة على القاعدة القانونية هو بوضوح (2).

❖ تعريف 2:

لتكن $\alpha=(V,E)$ دارة منطقية بـــ n دخلاً و m خرجاً ، يرمز لحجم الدارة α بـــ $Z(\alpha)$ وهــو عــدد العقد في الدارة أي إن: $Z(\alpha) = |V|$. إن $Z(\alpha) \geq n + m$.

يرمز بـــ $D(\alpha)$ لعمق الدارة α وهو الطول الأعظمي للمسالك في الدارة (أي بين جميــع المســالك الواصلة بين عقد الدخل وعقد الخرج).إن الحجم والعمق هما أهم مقياسين للتعقيد في الدارات المنطقية.

يظهر الشكل (1-10) دارة منطقية بسيطة فيها أربعة مداخل ومخرج واحد. حجمها وعمقهـا علـى التوالي هما 15 و 6.



شكل (1−10) يبين دارة منطقية بسيطة تعتمد على القاعدة القانونية فيها $Z(\alpha)=15$ و $D(\alpha)=6$

سنفترض فيما يلي أن كل عقدة دخل تتصل بعقدة خرج من خلال مسلك ، إذ إن كل دخل يستعمل في تحديد قيمة الخرج إلى حد ما مما يوضح أهمية العمق لدارة. إن الدارة α بـــ n دخـــلاً و m خرجـــاً حيث $n>m$ تملك عمقاً لا يقل عن $\lceil \log_t(n\backslash m)\rceil+1$ حيث t هو fan-in الدارة.

ويمكن أن يكون حساب الدالة لكل دارة ومن أجل ذلك سنضع بعض الملاحظات التمهيدية. لتكن الدارة α بـــ n دخلاً و m خرجاً ولتكن $\wedge$ ترمز إلى عنوان دالة في عقد α وهي:

$$\wedge : \{1,\ldots,Z(\alpha)\} \rightarrow \{x_1,\ldots,x_n\} \cup \{y_1,\ldots,y_m\} \cup \{0,1\} \cup \{f_1,\ldots,f_r\}$$

يعرف الدخل للدارة α بـــ n دخلاً بتحديد قيم المجموعة $\{0,1\}$ إلى المتغيرات المعنونة لعقد الدخل.

وبشكل نظامي فإن الدخل γ هو دالة معرفة على $\{x_1,\ldots,x_n\}$ بالقيم $\{0,1\}$:

$$\gamma : \{x_1,\ldots,x_n\} \rightarrow \{0,1\}$$

لنفترض أيضاً أن العناوين 0 و 1 ترمز إلى القيم 0 و 1 على التوالي.

القيم $V(k, \gamma)$ تحسب بواسطة العقد k على الدخل γ وتعرف كما يلي:

$$V(k,\gamma)=\begin{cases}\gamma(\wedge(k)) & \text{if } 1 \le k \le n \\ 0(\text{resp.}1) & \text{if } n+1 \le k \le n+c \text{ and } \wedge(k)=0 \text{ (resp.1)} \\ f_{l(k)}(V(i_{j1},\gamma),\ldots,V(i_{jk},\gamma)) & \text{if } n+c+1 \le k \le Z(\alpha)-m \,; \\ V(j,\gamma) & \text{if } k > Z(\alpha)-m \end{cases}$$

حيث: $f_{l(k)}$ عنوان العقدة k ويوجد في الغراف كل من الأقواس $(i_{j1},k), \ldots, (i_{jk},k)$

(في الحالة الأخيرة) ويوجد القوس (j,k)

أي أنها تحسب القيم كما يلي:

1- عن طريق عقدة الدخل: تدل القيمة على العنوان الخاص بالدخل γ.

2- عن طريق العقدة الثابتة: هي القيمة التي ترمز للعنوان.

3- عن طريق عقدة العمليات: توجد بتطبيق دالة العنوان على القيم المحسوبة بواسطة الأسـلاف الوسطى تبعاً للترتيب الثابت.

4- عن طريق عقدة الخرج: هي القيمة المعطاة بواسطة الأسلاف الوسطى فقط.

❖ **تعريف 3:**

القيمة المحسوبة بوساطة دارة α بـ m خرجاً للدخل γ هي السلسلة $Y(\alpha,\gamma) \in \{0,1\}^m$ تحسب على الطريقة النظامية بوساطة عقد الخرج $Z(\alpha)-m+1 \ldots (Z(\alpha)$ أي أن:

$$Y(\alpha,\gamma) = V(Z(\alpha)-m+1, \gamma) \ldots V(Z(\alpha),\gamma)$$

❖ **تعريف 4:**

تقوم الدارة α بـ n دخلاً و m خرجاً بحساب الدالة $f : \{0,1\}^n \rightarrow \{0,1\}^m$ ، من أجل أي دخـل γ فإن : $Y(\alpha,\gamma)= f(x_1, x_2, \ldots, x_n)$

نلاحظ أن الدارة تحسب دالة واحدة وواحدة فقط أي لها سلوك حتمي deterministic.

❖ تعريف 5: (إضافة خانة واحدة one bit adder)

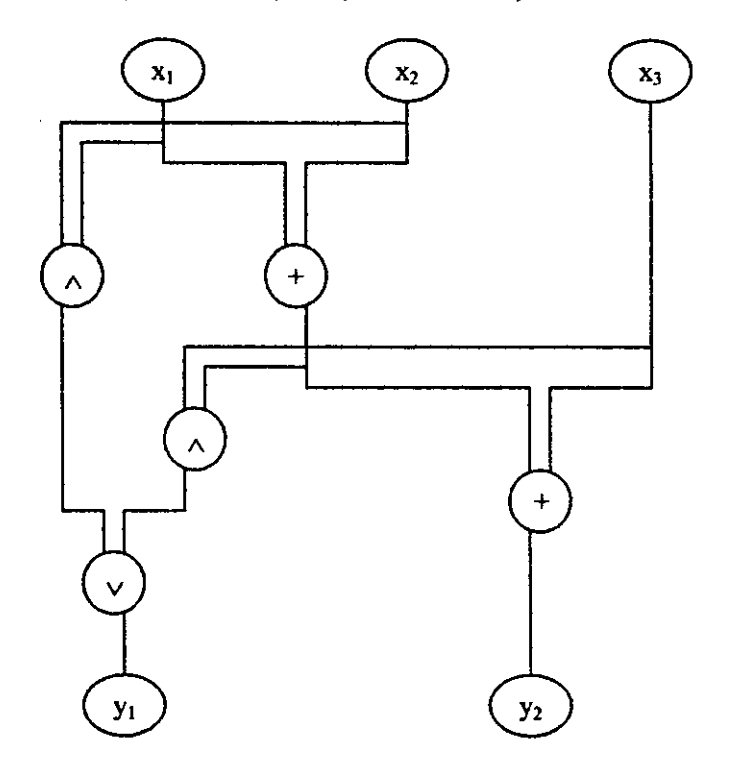
يبين الشكل (1-11) دارة تحسب الدالة:

$$f(x_1 , x_2 , x_3) = < Y_1 , Y_2 >$$

حيث:

$$Y_1 = f(x_1 , x_2 , x_3) = (x_1 + x_2 + x_3) \text{ div } 2$$

$$Y_2 = f(x_1 , x_2 , x_3) = (x_1 + x_2 + x_3) \text{ mod } 2$$



الشكل (1-11) إضافة خانة واحدة

تتألف الدارة من دارتين جزئيتين تحسبان OR المخصصة لخانتين ولجعل المخطط مقروء بشكل أوضح سنستعمل الاتفاقات التالية:

- سنضع الأضلاع عوضاً عن الأقواس ونحدد الاتجاه من القمة إلى القاعدة.
- يخرج من العقدة ضلع واحد فقط بينما نستعمل نقاط اتصال محتملة ليتفرع الضلع للوصول إلى العقد الخلَف.

نلاحظ أن fan-out لجميع عقد الدخل في الشكل السابق هو اثنان ولا واحد.

إذا كان لدينا مثلاً مسألة R لها الحجم n (ونرمز لذلك عادة بالرمز R_n) موصوفاً بـــ $g(n)$ خانة وإذا كانت حلول الأمثلة لـــ R_n موصوفة بـــ $h(n)$ خانة فإن الدارة المنطقية التي تحل R_n يجب أن تملـــك $g(n)$ مدخلاً و $h(n)$ مخرجاً.

❖ **تعريف 6:**

لتكن $R_n \subseteq \{0,1\}^{g(n)} \times \{0,1\}^{h(n)}$ ، تعتبر الدارة α ذات الـــ $g(n)$ مدخلاً والـــ $h(n)$ مخرجاً حلاً للمسألة المعرفة بـــ R_n إذا كانت تحسب الدالة :

$$f_n : \{0,1\}^{g(n)} \rightarrow \{0,1\}^{h(n)}$$

وإذا حققت : أنه إذا رمزت x لفرضية ما لـــ R_n وكانت x تملك على الأقل حلاً وحيداً فإننا نحصـــل على $(x, f_n(x)) \in R_n$.

يمكن أن تستعمل الدارات بمخرج واحد لتمييز اللغة على الأبجدية $\{0,1\} = \Sigma$ ، وبشكل دقيق أكثـــر إذا كانت A لغة ما معرفة على Σ (أي أن $A \subseteq \Sigma^*$) ، ولتكن A^n ترمز للغة $A \cap \Sigma^n$ وتتـــألف من السلاسل في A بطول n تماماً ، عندئذ ستستعمل الدارة ذات الـــ n إدخال $x_1,x_2,..,x_n$ والخرج الوحيد y لتمييز اللغة A^n إذا كانت $y=1$ وإذا كانت $x_1,x_2,..,x_n \in A^n$.

مثال(1-4):

الدارة المنطقية التي تظهر في الشكل (1-10) تميز سلاسل $w \in \{0,1\}^4 \cap 0^*1^*$ أي إن: السلاسل بطول 4 حيث جميعها أصفار إذا كان أي منها يسبق بـــ 1.

على سبيل المثال: السلسلة "0011" يمكن أن تميزها الدارة (وتكون قيمة الخرج $y_1=1$) بينما السلسلة "0101" ستستبعدها الدارة (أي أن $y_1= 0$).

لا يمكن اعتبار الدارة المفردة نموذجاً حسابياً مناسباً والسبب أن بنيتها الثابتة تمنع حل المسائل من أي حجم ولذلك نحن بحاجة أن نأخذ مجموعة من الدارات أفضل من دارة مفردة لكل حجم مفروض. ومن أجل ذلك لنأخذ التعريف التالي:

❖ تعريف 7:

تكون $\{\alpha_n\}_{n\in N}$ مجموعة دارات تحل المسألة R إذا وفقط إذا كان من أجل أي $n\in N$ فإن α_n تحل R_n

.

يمكن أن نوسع مفهومي الحجم size والعمق depth على مجموعات الدارات.

حجم (عمق) مجموعة $\{\alpha_n\}_{n\in N}$ هو دالة :

$Z_\alpha : N \to N \quad (D_\alpha: N \to N)$

بحيث: α_n لها الحجم $Z_\alpha(n)$ (العمق $D_\alpha(n)$) ، ونكتب $Z(n)$ و $D(n)$ ببساطة دون ذكر α.

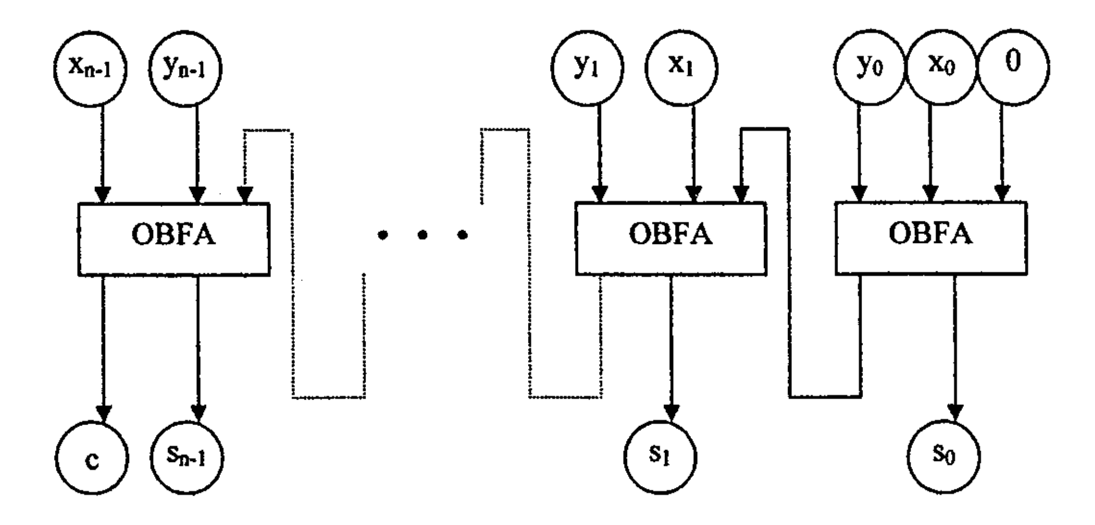
مثال(1-5) -إضافة (n-bit) n خانة- :

يمكن أن نحسب مجموع عددين صحيحين بـ n-bit $X = x_{n-1},\ldots,x_0$ و $Y= y_{n-1},\ldots,y_0$ عن طريق الدارة الموضحة في الشكل (1-12) حيث كل مقطع مسمى OBFA (موجود من أجل إضافة وإملاء one bit خانة واحدة "One-Bit Full Adder") هو جزء عملي operation part (أي أنه يحوي العمليات فقط) لإضافة خانة واحدة one-bit.

لاحظ أنه : تعنون في هذه الحالة عقد الدخل بـ x_i و y_i بينما تعنون عقد الخرج بـ s_i (حيث $i=0,\ldots,n-1$) و c .

الأقواس الداخلة لـ s_i حيث (حيث $i=0,\ldots,n-1$) تأتي من المحدد الثاني OR للمقطع OBFA (بمعنى آخر أنها الدخل للعقد المعنونة Y_2) بينما القوس الموجود في أقصى اليمين الذي يدخل إلى كل نموذج OBFA ما عدا الأول منها يأتي من عقد OR للـ OBFA السابقة (أي الدخل للعقدة المعنونة Y_1).

بما أن n عشوائية فإن الدارة في الشكل (1-12) تدل على مجموعة بالحجم والعمق 16n+2 و 2n+5 على التوالي.



الشكل (1-12) يمثل إضافة n خانة

8- ملاحظات:

سننهي هذا المقطع بملاحظات على مدلولات الحجم والعمق كمقياسين لتعقيد الـدارات. مـن الناحيـة العملية نجد أن حساب الدارة المنطقية للدالة f تصف الحسابات بشكل نظامي مختصر وهذه الحسابات هي التي تنفذها الخوارزمية التي تحسب الدالة f على حاسب حقيقي. وهذا ممكن على أية آلة حساب حقيقية على الأقل من أجل تحديد العناصر الحسابية الفعالة وترتيبها وفقاً للفعالية. ومع هذه المعلومات أصبح من الممكن بناء بيان graph يصف الحسابات بالطريقة التالية:

1- ندل على كل عنصر حسابي بواسطة عقدة.
2- إذا كان العنصر x عند الحساب يعطي عنصر بيانات خرج إلى العنصر y عندئـذ يُضـاف القوس (x,y) إلى البيان.

وباختصار إن الدارة المنطقية (البوليانية) أساساً هي اختصار لبيان حسابي مبني وفقاً لما ذُكِر. ومـن هنا نرى أن عقد العمليات هي عناصر حسابية بينما الأقواس هي وصلات لنقل البيانات مـن وإلـى الوسط الخارجي على التوالي.

بناء على وجهة النظر العملية هذه يتضح أن حجم الدارة هو تقدير مناسب لكلفة المكونـات الصـلبة لحساباته.

في الحقيقة إن حجم الدارة المنطقية يتعلق بشكل وثيق بعدد العناصر الحسابية في البيان المماثل. ومن جهة أخرى يمكننا أن ندرك أن الحجم عموماً هو تخمين سيء للكلفة. وهذا يعود لحقيقـة أن الـدارة المنطقية حلقية مما يجعل من المستحيل استعمالها للعقدة نفسها أكثر من مرة.

مقياس المكونات الصلبة البديل هو عرض الدارة (circuit width) نفترض لتعريفه أن جميع العقد في الدارة يمكن أن تقسم إلى مستويات وفقاً لما يلي:

1- عقد الدخل والعقد الثابتة هي المستوى 0 بالتعريف (level 0).
2- كل العقد الأخرى هي للمستوى i إذا وفقط إذا كان سلفها الوسطي في المستوى 0 أو في المستوى i-1 (تدعى الدارات التي تحدد هذا الشرط بـ synchronous).

عندئذ يمكننا أن نعرف عرض الدارة بأنه العدد الأعظمي للعقد (maximum) في كل مستوى.

يقيس العرض الأعظمي العناصر الحسابية العاملة بالتوازي في الحساب.

عمق الدارة depth يُحسب بواسطة دالة f هي من جهة أخرى قياس للزمن الكافي لحساب f.

يعتبر العمق مقياساً جيداً للزمن المتوازي. ولمعرفة ذلك يمكن أن نفكر بالتشابه مع الحاسوب الحقيقي.

نصف الدارة التي تنجز الحساب جيداً بوساطة زمن المعالجة المتناسب مع العدد الأعظمي للعناصر بغض النظر عن الإشارة حيث أننا نحد من المعالجة لعناصر الحساب المختلفة عن الصفر. وهذا يعني عمق الدارة نفسه.

نلاحظ أن الزمن هو حقاً زمن متوازٍ حيث إن تأثير العقد على المسالك يربط المداخل بالخرج وذلك يحدث على التوازي.

الفصل الثاني

دراسة الخوارزميات المتوازية وأنواعها وتمييز خواصها وتقنيات تصميمها ومقاييس تحليلها وتحديد كفاءتها

2-1 مقدمة عن الخوارزميات المتوازية:

منذ بداية عصر معالجة المعلومات أدرك العالم أن من المفيد جعل الأجزاء المختلفة للحاسوب تقوم بأشياء مختلفة في الوقت نفسه. فبينما تقوم وحدة المعالجة المركزية بالحساب يمكن أن تقرأ المدخلات من وسائط الدخل المتاحة وتخرج المخرجات على وسائط الخرج المتاحة.

وفي أجهزة متقدمة أكثر تحوي معالجات بسيطة متعددة يختص كل منها بعمل معين وتشترك فيما بينها لإنجاز المهام المقدمة للمعالجة.

وبالرغم من أن سرعة المعالجات قد ازدادت بشكل كبير إلا أن الطلب على معالجة أسرع للمعطيات أصبح ضروري أكثر كما أنه من غير الممكن في أغلب حالات تطبيقات الزمن الحقيقي إنجاز كل الحسابات باستخدام معالج واحد. وقد قاد ظهور الأجهزة المتوازية ذات المعالجات المتعددة المتزامنة وغير المتزامنة وظهور الشبكات الكبيرة إلى إعادة التفكير في مفهوم الخوارزميات إذ أن الخوارزمية تهدف بعد أن تتحقق من وجود الحل إلى البحث عن حل عملي.

كما أن الحاسوب المتوازي الذي يحوي العديد من وحدات المعالجة يقوم بتجزئة المسألة المعطاة إلى مسائل جزئية يقوم بحلها بشكل متزامن كل مسألة جزئية في معالج مختلف وبعد ذلك تجمع النتائج لإعطاء حل المسألة الأصلية. ويعتبر هذا انتقالاً جذرياً من نموذج الحساب بالحاسوب المعتمد في بناء الحاسبات أي ذات المعالجات التسلسلية. وقد أصبح التوازي يضاهي تحقيق سرعات عالية جداً للحساب بوساطة الحاسوب. ومع ظهور الحواسيب المتوازية وتوفر مكوناتها المادية نجد أن هناك سؤالاً مطروحاً وهو: كيف نبرمج الحواسيب المتوازية لحل المسائل بشكل فعال وعملي وبطريقة ذات كلفة منخفضة. بالطبع نحن بحاجة في الحساب المتوازي إلى خوارزميات ولغات برمجة ومترجمات ... الخ، بالإضافة إلى أن الأنظمة العملية بحاجة فعلاً لإنجاز الحساب على المكونات المادية المتوازية.

2-2 تعريف الخوارزمية المتوازية:

تعرف الخوارزمية المتوازية بأنها مجموعة من العمليات التي تنجز كل منها خوارزمية تتابعية خاصة بها وتتبادل الاتصالات فيما بينها عبر شبكة اتصالات محددة.

ويمكن أن نميز بين حالتين في الخوارزميات المتوازية وذلك حسب طريقة اتصال المعالجات فيما بينها إذ أنه في حين اتصال المعالجات عن طريق شبكة اتصالات محددة فإن طبولوجيا هذه الشبكة يعتبر

هاماً ومؤثراً في زمن تنفيذ هذه الخوارزمية وفي فعالية الخوارزمية وذلك تبعاً لوجود خط اتصال بين المعالجين الموافقين. وعند اتصال المعالجات بوساطة الذاكرة المشتركة القابلة للتقسيم يستطيع كل معالج الاتصال مع الآخر بدون كلفة زمنية ولا نهتم هنا بالطريقة التي تتم بها الاتصالات بين المعالجات.

وبما أن الخوارزمية المتوازية هي طريقة حل للمسائل على الحواسيب المتوازية. لذلك فهي تتبع بشكل أساسي للنماذج الرياضية الأساسية في الحاسوب المتوازي والتي قمنا بتعريفها في الفصل الأول وتتمايز أنواعها حسب أنواع الحواسيب المتوازية.

2-3 تحليل الخوارزميات المتوازية:

لقد ازدادت سرعة الحواسيب كثيراً في الأربعين سنة السابقة فقد كان يُعتقد أن فعالية الخوارزميات ليست ذات أهمية كبيرة ولكن الحقيقة التي ظهرت اليوم أن الفعالية أمر مهم أكثر مما سبق.

وهذا ما يدعونا إلى التعمق بتحليل الخوارزميات المتوازية لمعرفة فعاليتها،[JeHk-1997] وأهم أسباب فعاليتها هو أن الزمن الذي تأخذه معظم الخوارزميات للتنفيذ هو دالة غير خطية في حجم إدخالها وهذا يمكنه أن ينتج بشكل أكبر قدرتها على الإفادة في زيادة السرعة. كما أن الحواسيب السريعة تحتاج إلى خوارزميات أكبر للاستفادة من طاقتها.

يحدد تحليل الخوارزميات المتوازية درجة جودة الخوارزمية وهذا يعني سرعتها وكلفة تنفيذها ومدى فعاليتها عند استعمالها في الوسائل المتاحة. ولذلك فإننا نهتم بالمعايير التالية: زمن التنفيذ وعدد المعالجات المستخدمة ومن ثم نقوم بحساب الكلفة. ,[SeGA-1989] , [AnJe,1974] [GSAG,1994]

2-3-1 أولاً: زمن التنفيذ Running time

بما أن سرعة الحسابات هي السبب الرئيسي الذي جعلنا نهتم ببناء الحواسيب المتوازية لذلك فإن المقياس الأهم في تقييم الخوارزمية المتوازية هو زمن تنفيذها.

ويمكن تعريفه بأنه الزمن الذي تأخذه الخوارزمية خلال حل المسألة على حاسب متوازي. وذلك يعني أنه الزمن المستهلك من الخوارزمية منذ اللحظة التي تبدأ فيها إلى اللحظة التي تنتهي فيها.

إذا كانت المعالجات المتعددة لا تبدأ وتنتهي جميعها من حساباتها بنفس الوقت عندها فإن زمن التنفيذ يساوي إلى الزمن المستهلك بين اللحظة التي يبدأ فيها المعالج الأول ببدء الحساب واللحظة التي ينتهي فيها المعالج الأخير من الحساب.

المعايير المستخدمة في تحديد زمن التنفيذ:

- **1- خطوات العد Counting Steps**

في الواقع قبل تنفيذ الخوارزمية (سواء التسلسلية أو المتوازية) على حاسب ما فإنه من المعتاد توجيه التحليل النظري للزمن الذي تطلبه لحل المسألة الحسابية من جانب. وعادة يكون هذا عن طريق عدّ عدد العمليات الأساسية أو الخطوات المنفذة من الخوارزمية في أسوأ الأحوال.

ويمكن وصف هذه الخطوات بدالة حجم الإدخال. وبالطبع يختلف تعريف الخطوة من نموذج تقني إلى آخر.

وبشكل بديهي فإن عمليات المقارنة والإضافة ومبادلة رقمين هي عمليات أساسية مقبولة عموماً في معظم النماذج. كل من هذه العمليات يتطلب عدد ثابت من وحدات أو دورات الزمن في نموذج الحواسيب SISD (الذي يحوي معالج وحيد).

يمكن الحصول على زمن تنفيذ خوارزمية التوازي بِعَدّ نوعين من الخطوات: خطوات الحساب وخطوات الاضطراب أو الدوران Routing steps.

خطوة الحساب هي عملية رياضية أو منطقية تنفذ على عنصر بيانات في المعالج.

من ناحية أخرى فإنه في خطوة الاضطراب ينتقل عنصر البيانات من أحد المعالجات إلى آخر عن طريق الذاكرة المقسمة أو عن طريق شبكة الاتصالات.

من أجل مسألة بحجم n فإن زمن تنفيذ أسوأ حالة توازي في خوارزمية هو دالة في n سنرمز لها بـ t(n). كما أن زمن التنفيذ هو أيضاً دالة لعدد المعالجات.

عموماً الخطوات الحسابية والخطوات الاضطرابية لا تتطلب بالضرورة نفس عدد وحدات الزمن حيث أن خطوة الاضطراب عادة تعتمد على المسافة بين المعالجات ونموذجياً تأخذ زمن تنفيذ أقل طولاً من الحسابية.

• 2- الحدود الدنيا والعليا LOWER AND UPPER BOUNDS

من الشائع بين مصممي الخوارزميات في مسألة حسابية مصممة فقط من أجل خوارزمية تسلسلية جديدة تطبيق السؤالين التاليين:

1- هل هذه هي الخوارزمية المحتملة الأسرع للمسألة ؟

2- إذا لم تكن كذلك كيف يمكن مقارنتها مع الخوارزميات الأخرى الموجودة لنفس المسألة ؟

تحصل الإجابة عن السؤال الأول عادة عن طريق مقارنة عدد الخطوات المنفذة من الخوارزمية لمعرفة الحد الأدنى لرقم الخطوات المطلوبة لحل المسألة في أسوأ حالة.

مثال(2-1):

لنقل أننا نرغب بحساب حاصل ضرب مصفوفتين $(n \times n)$. بما أن المصفوفة تتطلب n^2 إدخال على الأقل فإن هذه الخطوات العديدة مطلوبة من أي خوارزمية ضرب مصفوفات لإنتاج الخرج. الحدود الدنيا في المثال 2-1 تُعرف بالحدود الدنيا البدائية أو الواضحة والتي يمكن الحصول عليها من عدّ عدد الخطوات المطلوبة خلال الدخل و/أو الخرج.

الحدود الدنيا المتطورة أكثر مذكورة في المثال 2-2:

مثال(2-2):

تعرف مسألة الترتيب كالتالي: لنأخذ مجموعة من n رقم مرتبة عشوائياً. رتب هذه الأعداد تنازلياً. يوجد هنا !n تبديل محتمل للإدخالات و log n! بت مطلوبة للتمييز فيما بينها. لذلك فإن أي خوارزمية ترتيب في أسوأ حالة تتطلب n log n خطوة على الأقل لتمييز الخرج الخاص.

إذا كان عدد خطوات الخوارزمية المنفذة في أسوأ حالة مساوي للحد الأدنى عندها تكون هي الاحتمال الأسرع (الأفضل) وإلا سنكون بحاجة لخوارزمية تحسن الحد الأدنى. وعلى أي حال إذا كانت الخوارزمية الجديدة أسرع من جميع الخوارزميات المعروفة للمسألة فإننا نقول أنها تؤسس الحد الأعلى الجديد على عدد الخطوات المطلوبة لحل تلك المسألة في أسوأ حالة.

أما عن السؤال الثاني فيكون دائماً عن طريق مقارنة زمن التنفيذ للخوارزمية الجديدة مع الحد الأعلى الموجود للمسألة .

مثال(2-3):

حتى هذا التاريخ لا يوجد أي خوارزمية معروفة لضـــرب مصـــفوفتين $(n \times n)$ فـــي n^2 خطـــوة. والخوارزمية القياسية المدروسة تحتاج n^3 عملية. ويميز الحد الأعلى لهـــذه المســـألة عـــن طريـــق خوارزمية تتطلب n^x خطوة على الأغلب حيث $x<2.5$.

وعلى سبيل المقارنة يوجد خوارزميات ترتيب متعددة تتطلب على الأغلب $n \log n$ عملية ولذلك هي الأفضل.

إن هذه المعالجة للحدود الدنيا والعليا مركزة على الخوارزميات التسلسلية. وهذه الأفكار العامة نفســـها تطبق على الخوارزميات المتوازية ولكننا نأخذ عاملين إضافيين في الاعتبار:

1- نموذج الحسابات المتوازي المستعمل
2- عدد المعالجات المستخدمة

- **3- زيادة السرعة Speedup**

عند تقييم خوارزمية التوازي لمسألة معطاة فإنه من الطبيعي تماماً فعل ذلك بمدة الخوارزمية التسلسلية المتوفرة الأفضل لتلك المسألة. ولذلك فإنه يوجد دلالة جيدة لمراقبة نوعية quality خوارزمية التوازي وهي زيادة السرعة التي تنتجها:

زيادة السرعة = X/Y ؛ حيث:

X = زمن تنفيذ أسوأ حالة للخوارزمية التسلسلية المعروفة الأسرع للمسألة

Y = زمن تنفيذ أسوأ حالة لخوارزمية التوازي

ومن الواضح أن زيادة السرعة الأكبر و الأفضل هي خوارزمية التوازي.

2-3-2 ثانياً: عدد المعالجات

المعيار الثاني الأهم في تقييم الخوارزمية المتوازية هو عدد المعالجات المطلوبة في حل مسألة. حيث أنها مكلفة من أجل الشراء والصيانة وتشغيل الحواسيب.

عندما يوجد عدة معالجات تنشأ مسألة الصيانة بشكل خاص وكذلك يرتفع السعر المدفوع بشكل حاد من أجل ضمان درجة عالية من الوثوقية. لذلك فإن العدد الأكبر من المعالجات في الخوارزمية المطلوبة لحل مسألة يجعل الحل أكثر غلاءً.

من أجل مسألة من الحجم n يكون عدد المعالجات المطلوبة في الخوارزمية هو دالة بـ n ولتكن p(n). وفي بعض الأحيان يكون عدد المعالجات هو ثابت يعتمد على n.

3-3-2 ثالثاً: حساب الكلفة ACCOUNT OF COST

تعرف كلفة خوارزمية التوازي على أنها حاصل ضرب المعيارين السابقين:

الكلفة = زمن التنفيذ × عدد المعالجات المستخدمة

وبكلمات أخرى : الكلفة تساوي عدد الخطوات المنفذة بشكل جماعي من قبل المعالجات في حل مسألة بأسوأ حالة. يفترض التعريف أن جميع المعالجات تنفذ عدد الخطوات نفسه.

إذا لم تكن الحالة كذلك فإن الكلفة هي الحد الأعلى في ناتج عدد الخطوات المنفذة.

من أجل مسألة حجمها n فإن كلفة خوارزمية التوازي هي دالة في n نرمز لها بـ c(n) ومنه:

$$c(n)=p(n).t(n)$$

- ويمكن أن نأخذ بعين الاعتبار وجود معايير أخرى في حساب كلفة الخوارزميات:

يمكن أن تعرض الحواسيب الرقمية كمجموعات كبيرة من بوابات الاتصال المنطقية وهذه البوابات تبنى باستعمال الترانزستورات والمقاومات والمكثفات (أو الدارات التكاملية والتفاضلية). واليوم تأتي البوابات في حُزَم تدعى رقاقات chips وهي قطع صغيرة جداً من مادة شبه ناقلة تستعمل لصنع البوابات المنطقية والأسلاك التي تصل بينها. عدد البوابات على الرقاقة يحدد مستوى الدمج الموجود المستعمل لبناء الدارة. إن أحد التقنيات الخاصة التي ظهرت في الاتصالات بنجاح مستقبلياً في الحساب المتوازي هو: مقياس الضم الكبير جداً very large scale integration (VLSI) وفيه يمكن أن تتوضع مليون بوابة منطقية تقريباً على رقاقة مفردة $1cm^2$ ولذلك فإن الرقاقة قادرة على أن تشمل عدد من المعالجات ويمكن أن تجتمع العديد من مثل هذه الرقاقات لبناء حاسب متوازي قوي.

وعند تقييم الخوارزميات المتوازية في مقياس الـ VLSI غالباً ما تستخدم المعايير التالية:

1. مجال المعالج processor area
2. طول السلك wire length
3. مهلة الدارة period of the circuit

2-4 تصميم الخوارزميات المتوازية:

إن عملية تصميم خوارزمية متوازية هي عملية حركية حيوية ولا يوجد قاعدة معينة للحصول على خوارزمية متوازية ذات كلفة أصغرية . كما أن هناك العديد من المسائل الهامة لم يتم إيجاد خوارزميات متوازية ذات كلفة معقولة لحلها ، ولكن هناك بعض الاستراتيجيات الرئيسية تقود للحصول على خوارزميات ذات كلفة مقبولة. [IFos,1994], [JeHk-1997], [BrCMal_1992], [JHKi,1990]

إن الشكل الطبيعي لتطوير الخوارزميات المتوازية هو بالبدء من الخوارزمية التسلسلية ومن ثم تحديد الخطوات المستقلة وأخيراً الحصول على الخوارزمية المتوازية. وهذا التحول من التسلسل إلى التوازي ينتج غالباً خوارزميات متوازية ذات كلفة منخفضة. ومن أجل ذلك تنشأ مسألة تحديد استراتيجيات للحصول على خوارزميات متوازية فعالة حيث لا يوجد طريقة عامة تعطي أفضل خوارزمية متوازية لمسألة معطاة ولكن هناك تقانات تستعمل لهذا الغرض وسنعرض بعض هذه الطرائق والتقانات:

سنرمز فيما يلي إلى كل من الزمن والمكونات المادية وعدد المعالجات وللزمن ولعدد المعالجات بـأن معاً بـ: T و H و P و PT على التوالي.

1. التقسيم والتخصيص Divide et impera:

تعتمد هذه التقانة على تقسيم المسألة إلى مسائل جزئية من نفس النوع ولكن بحجم أصغر ومن ثم إعادة ترتيب وتجميع الحلول. وهي طريقة فعالة جداً حيث تسمح بتجزئة المسألة إلى مسائل جزئية مستقلة تُحل على التوازي وتُجمع حلولها للحصول على حل المسألة الاصلية.

أما بالنسبة لكلفة الحل المتوازي لمسألة حسابية من الترتيب n وبالتالي من الحجم $H(n)$ والعمق $T(n)$ لدينا:

لتكن r عدد صحيح يقسم n ولتكن $H_i(n/r)$ و $T_i(n/r)$ (حيث: $i=1,2,..,r$) هي كلف حل r مسألة جزئية ، ولتكن $T_c(n)$ و $H_c(n)$ هي الكلفة المتوازية الإضافية لحل المسألة الأصلية بعد أن يتم حل المسائل الجزئية. لدينا:

$$T(n) \le \max(T_1(n/r), T_2(n/r), \ldots, T_r(n/r)) + T_c(n) \quad \ldots(1)$$

$$H(n) \le \max(H_1(n/r) + H_2(n/r) + \ldots + H_r(n/r), H_c(n)) \quad \ldots(2)$$

تأتي العلاقة (1) من الحقيقتين التاليتين :

1- استقلالية المسائل .

2- الحاجة لتنفيذ العمل الإضافي عندما تُحل المسائل الجزئية فقط.

تأتي العلاقة (2) من الحقيقتين التاليتين:

1- كل مسألة جزئية تم حلها على التوازي مع الأخريات تحتاج مصادرها المادية الخاصة من المكونات الصلبة hardware.

2- الكمية الناتجة من المكونات الصلبة هي العدد الأعظمي بين المصادر المطلوبة لحل جميع المسائل الجزئية والمصادر المطلوبة للحصول على الحل الأخير للمسألة الأصلية.

وعلى افتراض أن جميع المسائل الجزئية متساوية من حيث كلفة التوازي تصبح العلاقتين (1) و (2) كما يلي:

$$T(n) \le T(n/r) + T_c(n) \quad \ldots(3)$$

$$H(n) \le \max(rH(n/r), H_c(n)) \quad \ldots(4)$$

حيث $T(n/r)$ و $H(n/r)$ هي كلفة المسألة الجزئية من الترتيب n/r.

تعتبر هذه العلاقات من 1 إلى 4 هامة من أجل تحليل الخوارزميات المتوازية التي نحصل عليها من طريقة التقسيم والتخصيص.

2. طريقة استخدام المتجه Vectorization:

تبدأ هذه التقانة من تحليل المسألة وإنتاج الخوارزميات التي تعمل على بنى المعطيات التي تتناسب مع حسابات المتجه (المتجهات والمصفوفات). وهذه التقانة هامة بشكل خاص في النموذج SIMD خاصةً عندما يكون من المناسب العمل على متجه من n ترتيب بالإضافة إلى n كمية غير متجهة.

3. تكرارات المتجهات vector iterations:

قام بوضع هذه التقانة Traub وآخرون من أجل حل الجمل الخطية المثلثية. حيث تبدأ باستخدام الخوارزميات التسلسلية المباشرة وتبنى منها الطريقة التكرارية المتوازية وهكذا..

إذا أخذنا الطريقة التعاودية لحساب عناصر مصفوفة مثلثية بالشكل التالي:

$x_1 = a_1$
$x_i = a_i - (b_i c_i / x_{i-1}) \quad , 2 \le i \le n$

عندئذ يمكن أن نحصل على التكرارات التالية وفق طريقة غاوس:

$x_i^{(0)} = a_1$
$x_i^{(j)} = a_i - (b_i c_i / x_{i-1}^{(j-1)})$

عندما j هي رقم التكرار.

يمكن أن تكون هذه التكرارات شعاعية أي أنه يمكن أن نفترض تعديلاً للشعاع $x^{(j)}$ حيث: $j \ge 0$.

كما نجد أنه في الخطوة الأولى (j=0) كل عنصر في الشعاع $x^{(0)}$ تحوي القيمة a_1.

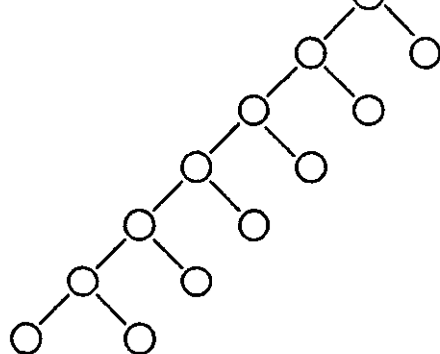
في الخطوة الثانية يحوي العنصر رقم i لـ $x^{(1)}$ القيمة:

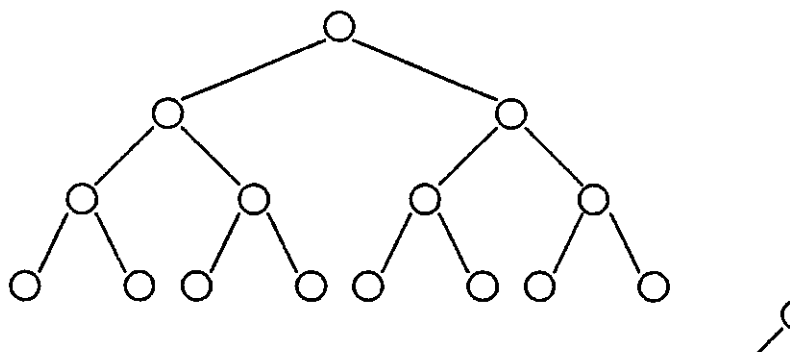
$a_i - (b_i c_i / x_{i-1}^{(0)}) = a_i - (b_i c_i / a_1) \quad , \quad i = 1,2,\ldots,n$

عندئذ يمكن أن تعدل كل العناصر في $x^{(1)}$ بآن واحد معاً. وهكذا يطبق المبدأ نفسه على التكرارات التسلسلية الجزئية.

4. المضاعفة التعاودية Recursive doubling:

وهي تتألف من تحويل البيان الحسابي من مثل البيان المعروض في الشكل (2-1 -أ-) إلى بيان من الشكل(2-1 -ب-) وهذا يمكننا من الحصول على بيان تكون فيه كلفة التجوال (درجة التعقيد) تابع أسي (لوغاريتمي) من بيان فيه كلفة التجوال (درجة التعقيد) تابع خطي.


الشكل(2-1) -أ- بيان بدرجة تعقيد خطية


الشكل(2-1) -ب- بيان بدرجة تعقيد لوغاريتمية

وفي هذا الشكل من أجل n=8 نلاحظ أن عمق البيان الحسابي في الشكل (2-1 -أ-) هو 7 وفي الشكل (2-1 -ب-) هو 3.

مثال(2-4):

لنأخذ حساب الـ x^n حيث $n=2^k$ من أجل عدد صحيح k . ولنأخذ الخوارزمية التالية:

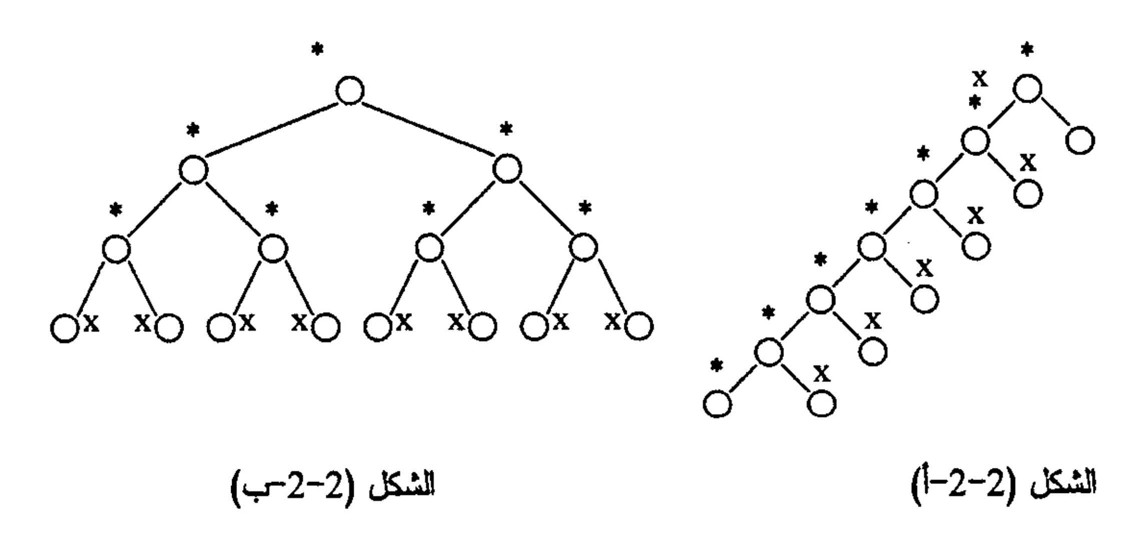
```
Begin
   s ← x;
   for i ← 1 until n-1 do
      s ← s*x;
end.
```

نلاحظ أن البيان الحسابي الناتج عن هذه الخوارزمية له درجة تعقيد خطية (أو عمق خطي) كما نجد ذلك موضحاً في الشكل (2-2-أ).

أما إذا كان الحساب يتم بالطريقة التالية:

```
Begin
   s ← x;
   for i ← 1 until k do
      s ← s*s ;
end.
```

عندئذ نحصل على البيان الموضح في الشكل (2- 2-ب) وهو بدرجة تعقيد لوغاريتمية.



الشكل (2-2-أ)   الشكل (2-2-ب)

يوضح الشكل (2-2-أ) بيان حسابي من أجل حساب تسلسلي لـ x^8 بدرجة تعقيد خطية ويوضح الشكل (2-2-ب) بيان حسابي من أجل حساب تسلسلي لـ x^8 بدرجة تعقيد لوغاريتمية

تملك البيانات الحسابية لخوارزميات تسلسلية من أجل حسابات رياضية عادة البنية الموضحة في الشكل (2-1-أ) ، ولا يتضح دائماً فيما إذا كانت الاستقلالية بين العمليات تعود للخوارزمية أم لأساس المسألة. ففي حالة اعتمادها على أساس المسألة لا يمكننا التغيير غالباً. أما في حالة اعتمادها على الخوارزمية عندئذ يمكننا أن نتوصل إلى تقريب للمضاعفة التعاودية الممكنة بتحليل دقيق للمسألة.

يمكن أن نعتبر أن المضاعفة التعاودية هي حالة خاصة من التقسيم والتخصيص. إذ أن البيان في الشكل (2-2- ب) يفترض أن القيمة x^n تساوي ناتج $x^{n/2} * x^{n/2}$ وهذا بدوره يعني أن المسألة p_n (أي الحساب لـ x^n) يمكن أن يحلل إلى $p_{n/2} * p_{n/2}$ أفضل من $x * p_{n/2}$. وبشكل أعم تتألف المضاعفة التعاودية من الحصول على خوارزمية متوازية بتكرارات بدءاً من الخوارزمية التسلسلية بكلفة s(n) تعطى بما يلي:

$$S(n) \leq S(n-r) + f(n)$$

حيث r ثابت و f(n) دالة متزايدة في n.

5. تخفيض عدد المعالجات Reducing the number of processor:

حيث يمكن في بعض الأحيان إنقاص عدد المعالجات الكبير باستعمال خوارزمية متوازية وذلك بزيادة زمن التنفيذ بواسطة عامل ثابت. ولنرى كيفية تحقق هذا الهدف. لنفترض أن لدينا خوارزمية متوازية تأخذ كلفة زمنية O(log n) وتستعمل O(n) معالج. ولنفترض أيضاً أن أفضل خوارزمية تسلسلية متوفرة تحل المسألة نفسها لها كلفة خطية. وهذا يؤدي إلى أن الخوارزمية المتوازية لها فعالية O(1/log n) وهي بعيدة عن الكلفة الأمثلية. في بعض الأحيان يمكن تحويل خوارزمية متوازية بمثل هذه الفعالية إلى خوارزمية متوازية أخرى بزمن O(log n) وعدد المعالجات O(n/log n) ومنه بفعالية θ(1) ، ومنه ينتج ما يلي: يجب أن تقسم المسألة من الحجم n كما في طريقة التقسيم والتخصيص إلى n/log n مسألة جزئية بحجم log n ، عندئذ تُحل كل مسألة جزئية بواسطة معالج مفرد باستعمال أفضل خوارزمية تسلسلية متوفرة (خطياً مع log n). إذا وصل حل كل من المسائل الجزئية الـ n/log n إلى المسألة الابتدائية من الترتيب n/log n عندئذ نستطيع أن نستعمل الخوارزمية المتوازية بأخذ زمن:

$$O(\log(n/\log n)) = O(\log n) \text{ مع } n/\log n \text{ معالج}$$

وبشكل عام يجب أن تطبق المعادلة التالية على المسألة p_n ، من أجل k يقسم n:

$$P_n = k\, P_{n/k} + P_k \qquad \ldots (5)$$

تعني العلاقة (5) أن المسألة p_n يمكن أن تجزأ إلى k مسألة جزئية $p_{n/k}$ من نفس النوع ، تولد حلولها مسألة p_k يمكن أن تُحل بعد أن يتم حل جميع المسائل الجزئية الــ k . لنفترض أنه لكي نحل p_n يوجد لدينا:

1- خوارزمية متوازية A_p تُمثل بــ $T_{Ap}(n)$ و $H_{Ap}(n)$

2- خوارزمية تسلسلية A مع كلفة (مثلاً عدد العمليات) $T_A(n)$

لنفترض أن $T_{Ap}(n)\, H_{Ap}(n) = T_A(n)\, f(n)$ حيث f(n) دالة متزايدة بــ n.

ولذلك فإن فعالية الخوارزمية A_p تعطى بالعلاقة 1/f(n) وهذا يعني أن الخوارزمية تقوم باستعمال إضافي (زائد) للمصادر.

ومنه يمكننا أن نستنتج ونفترض دالة g(n) كما يلي:

1. $\Omega(1) = g(n) < n$

2. $T_{Ap}(g(n))\, H_{Ap}(g(n)) = T_A(n)$

نستبدل k بــ g(n) في العلاقة (5) ونحصل على ما يلي:

$$P(n) = g(n)\, p_{n/g(n)} + p_{g(n)} \qquad \ldots (6)$$

من العلاقة (6) نجد أنه يمكن حل p_n كما يلي:

1. حل $p_{n/g(n)}$ بمعالج مفرد باستعمال خوارزمية A.

2. حل $p_{g(n)}$ باستعمال خوارزمية متوازية A_p .

يماثل هذا الإجراء خوارزمية متوازية جديدة *A تمثل بــ *T و *H تعطى بالعلاقة:

$$T^*(n) = T_{Ap}(g(n) + T_A(n/g(n)))$$
$$H^*(n) = \max \{ H_{Ap}(g(n)) , g(n) \}$$

عندئذ فإن فعالية *A هي O(1) مما ينتج المعادلات التالية المطبقة على g(n):

$$g(n) = O(H_{Ap}(g(n)))$$
$$T_{Ap}(g(n)) = O(T_{Ap}(g(n)))$$

5-2 كفاءة خوارزمية:

نعرف كفاءة أو كلفة خوارزمية بأنها ناتج ضرب عدد المعالجات التي تستعمل لإنجاز العملية وزمن التنفيذ للعملية.

[WaCDaK], [TLCP,1993] , [JHro,1997] إن حل الجمل الكبيرة من المعادلات الخطية يمكن أن يكلف كثيراً على الحاسب. وحتى نعرف لماذا دعنا ننجز عملية عد لخوارزمية لها كود معطى. سنعد فقط عمليات الضرب والقسمة (العمليات الطويلة) لأن لها أكبر زمن مستهلك، بعد ذلك نجمّع عمليات الضرب والقسمة معاً حتى ولو كانت عمليات القسمة أبطأ من عمليات الضرب.

مثال(5-2):

لنأخذ خوارزمية Gouss الذي نقوم فيه بحل جملة معادلات خطية ومن ثم نقوم بحساب الكلفة وذلك بعد العمليات الطويلة فيه، وفيه مصفوفة الأمثال $(a_{ij})_{n*n}$ ومصفوفة الثوابت $(l_i)_n$ وهناك مصفوفة مساعدة لتحديد مواقع العناصر هي $(s_i)_n$.

وفيما يلي نعرض خوارزمية $Gouss(n,(a_{ij}),(l_i))$ وفق التسلسل:

```
      Step1:
      for i=1 to n do
            li ← i
            smax ← 0
            for j=1 to n do
                                    smax ← max(smax , |aij|)
            end for
            si ← smax
      end for
step2:
for k=1  to  n-1 do
            rmax←0
      for i=k  to  n  do
                  r← |ali,k/sli|
                  if  (r> rmax) then
                        rmax ← r
                        j ← i
                  end if
      end for
      li  ↔lk
```

```
for i=k+1 to n do
        xmult←a_{li,k}/a_{lk,k}
        a_{li,k}← xmult
        for j=k+1 to n do
                a_{li,j}← a_{li,j}-(xmult)a_{lk,j}
        end for
    end for
end for
```

وفيما يلي نحسب كلفة التنفيذ:

في الخطوة 1: يتطلب اختيار العنصر المحوري حساب n نسبة وهذا يعني: n مقسوم. ومن ثم من أجل الأسطر $l_1,l_2,..,l_n$ نبدأ أولاً بحساب الضرب ومن ثم نطرح من السطر l_i مرات الضرب وهي السطر l_1. أما الصفر الذي ينشأ من هذه العملية فلا يحسب. ومنه يتطلب الحذف n-1 مضروب في كل سطر.

إذا ضمنّا حساب المضاريب فإننا نجد n-1 عملية طويلة (مقاسيم ومضاريب) في كل سطر. ونجد أن هناك n-1 سطر يجب معالجتها لنحصل بالنتيجة على n(n-1) عملية. إذا أضفنا كلفة حساب النسب فإن النتيجة هي n^2 عملية مطلوبة من أجل الخطوة1.

الخطوة التالية مثل الخطوة 1 ما عدا أن السطر l_1 لا يؤثر بل على العكس فإن عمود المضاريب قد نشأ وخُزّن في الخطوة1.

لذلك فإن الخطوة 2 ستتطلب $(n-1)^2$ مضروب ومقسوم تتم معالجتها على الجملة مع السطر l_1 بدون العمود1.

باستمرار هذه المعالجة نحصل على أن نتيجة عدد العمليات الطويلة للإجراء Gouse هي:

$$n^2+(n-1)^2+(n-2)^2+\ldots+4^2+3^2+2^2=n/6(n+1)(2n+1)-1\approx n^3/3$$

لاحظ أن عدد العمليات الطويلة في هذا الإجراء ينمو إلى $n^3/3$.

2-5-1 نظرية(2-1):

إن الحذف المتقدم المتجانس لخوارزمية حذف غاوس مع تخفيض المحاور الجزئية إذا طبقت فقط على مصفوفة معاملات n*n تحل تقريباً بـــ $n^3/3$ عملية طويلة (ضرب وقسمة). والحل من أجل x يحتاج n^2 عملية إضافية طويلة.

لنقوم الآن بإيجاد خوارزمية متوازية لمسألة حل جملة معادلات خطية وسنستخدم طريقة غاوس جوردان حيث أنها الطريقة التسلسلية الأكثر شيوعاً لحل هذه المسألة، وهي تتألف من حذف جميع المجاهيل x ما عدا x_i من المعادلة i وبذلك نحصل على الحل مباشرة.

[SeGA-1989] الخوارزمية المتوازية لطريقة غاوس جوردان مصممة لتنفذ على حاسب CREW SM SIMD بـ n^2+n معالج يفترض بأنها مرتبة في مصفوفة $n*(n+1)$. تعطى الخوارزمية كإجراء SIMD GAUSS JORDAN. ونرمز فيها لـ b_i بـ $a_{i,n+1}$.

2-5-2 الإجراء SIMD GAUSS JORDAN (A,b,x):

```
step1:
for j=1 to n do
        for i=1 to n do in parallel
                for k=j to n+1 do in parallel
                        if ( i ≠j) then
                                                          aik← aik – (aij/ajj)ajk
                        end if
                end for
        end for
end for
step2:
for i=1 to n do in parallel
        xi ← ai,n+1/aii
end for
```

نلاحظ أن الإجراء يسمح بعمليات القراءة المتعددة لأكثر من معالج واحد وذلك عندما يحتاج لقراءة a_{ij} و a_{jj} و a_{jk} بآن واحد.

- **حساب الكلفة التنفيذية:**

الخطوة 1 تتألف من n تكرار بزمن ثابت ، والخطوة 2 تأخذ زمناً ثابتاً. لذلك $t(n)=O(n)$. طالما أن $P(n)=O(n^2)$ فإن $C(n)=O(n^3)$. على الرغم من أن هذه الكلفة تصل إلى عدد الخطوات المطلوبة للخوارزمية التسلسلية لخوارزمية غاوس جوردان إلا أنها ليست أفضلية، وذلك لأن زمن التنفيذ الكلي للحل التسلسلي لجملة المعادلات الخطية $Ax=b$ هو $O(n^x)$ حيث $2<x<2.5$.

2-6 مقارنة بين الخوارزميات التسلسلية والمتوازية:

[JeHk,1997] , [RaRz-1993] يفيد تحليل الخوارزميات في مقارنة خوارزميات حل مسألة معينة فما نريد الوصول إليه هو مهما تكن الآلة التي ستنفذ الخوارزمية ومهما تكن لغة البرمجة المستخدمة فإن : لنفرض أن لدينا خوارزميتين A و B تنفذان المهمة البرمجية نفسها فإن الخوارزمية A أفضل من الخوارزمية B.

بعد تحديد تعقيد خوارزمية كتابع لحجم المعطيات يمكن دراسة سرعة تزايد هذا التابع عندما يزداد حجم المعطيات. تفيد هذه الدراسة في تحديد فعالية الخوارزمية من أجل معالجة معطيات كبيرة الحجم إذ يمكن في بعض الحالات أن نجد فروقاً هائلة بين خوارزميتين من حيث التعقيد الزمني.

في معظم الحالات نكتفي بتقريب بسيط لتابع التعقيد الزمني لمعرفة فعالية الخوارزمية ولمقارنة خوارزميتين. فمثلاً عندما تكون n كبيرة يكون من غير المهم معرفة ما إذا كانت خوارزمية معينة تحتاج لـ n أو n+5عملية. مما يدفعنا إلى إيجاد ما يسمى مرتبة كبر تابع حيث تتم مقارنة الخوارزميات على أساس مرتبة الكبر لتوابع التعقيد الزمني.

- **نتيجة (2-1):**

بفرض أن عدد المعالجات المستخدمة لإنجاز عملية في إحدى خوارزميات المعالجة المتوازية هو p وبفرض زمن تنفيذ الخوارزمية هو t ، وجدنا أن كلفة الخوارزمية C هي $C=t*P$:.
في الآلة التسلسلية عدد المعالجات واحد وكلفة الخوارزمية تساوي زمن التنفيذ تماماً ومنه $C=t*1$.
ومنه فإن الخوارزمية المتوازية بالكلفة الأمثلية هي الخوارزمية التي كلفتها :
O(t)=O(زمن التنفيذ التسلسلي)

- **نتيجة (2-2):**

إذا لم تكن الخوارزمية المتوازية ذات كلفة أمثلية فإنه يمكن تحسين الخوارزمية من خلال تخفيض زمن التنفيذ التقريبي أو تخفيض عدد المعالجات.

2-6-1 مفاهيم أساسية لبناء خوارزمية متوازية بكلفة أمثلية:

[ASSZ,1998] ليكن زمن تنفيذ الخوارزمية التسلسلية لإنجاز مهمة بحجم n هو O(n) وزمن تنفيذ الإصدار المتوازي لهذه الخوارزمية هو O(log n) حيث: n عدد المعالجات المستعملة، كل منها يمسك عنصر بيانات واحد. ولذلك فإن الكلفة هي:

$Cost = nO(\log n) = O(n\log n)$ هذه الكلفة ليست أمثلية. وللحصول على خوارزمية بكلفة أمثلية نحتاج أحد أمرين : إما تخفيض زمن التنفيذ O(1) أو إنقاص عدد المعالجات إلى n/logn ولذلك فإن ناتج الضرب لـ n/log n مع O(log n) يعطي O(n).

ليكن كل معالج يأخذ log n عنصر بيانات. فإذا كانت العملية التسلسلية خطية في عدد عناصر البيانات عندها كل من الـ n/log n معالج سينجز عملية تسلسلية على التوازي على كل من عناصر بياناتها الـ log n في زمن O(log n). إضافة إلى أن المعالجات الـ n/logn ستنجز الخوارزمية السابقة (غير الأفضلية) على النتائج n/log n من الخطوة التسلسلية في:

$$\log(n/\log n) = \log n - \log(\log n) = O(\log n)$$

ومنه فإن كلفة كل خطوة هي O(log n) والكلفة هي: $(n/\log n)O(\log n) = O(n)$ وهو المطلوب.

2-7 خواص الخوارزميات المتوازية: ٦٣٥٧٥٩

هناك خصائص هامة نرغب في الحصول عليها في الخوارزمية المتوازية.

1. ***عدد المعالجات:*** الخاصية الأولى هي عدد المعالجات المستخدمة في الخوارزمية. بفرض n هو حجم المسألة التي نريد حلها فإن:

أ- P(n) يجب أن تكون أصغر من n : إنه من غير الواقعي أن نفترض أن المعالجات أكثر من مفردات المعلومات خاصة من أجل قيم كبيرة جداً لـ n. ولذلك من المهم أن تكون P(n) تعبر عن دالة أسية أو لوغاريتمية في n وهذا يعني: $P(n)=n^x \; ; \; 0 < x < 1$.

ب- P(n) يجب أن تكون متكيفة adaptive: في الحساب بشكل عام وفي الحساب المتوازي بشكل خاص نحتاج إلى قوة حسابية إضافية إذ أنه دائماً يمكن أن تُطرح مسائل أكبر وأكثر تعقيداً مما كان محتملاً قبل ذلك. وهذه المسائل الأكبر قد تجعل الخوارزمية عديمة النفع تماماً. وتستخدم الخوارزميات عدد المعالجات على أنها دالة أسية أو لوغاريتمية في n

مثل $n^{1/2}$, log n والتي قد تكون غير مرضية بسبب عدم مرونتها. مـا نحتاجـه هـو خوارزمية تمتلك الذكاء لكي تتكيف مع عدد المعالجات الحقيقي (الواقعي) المتوفر علـى الحاسب الموجود.

2. ***زمن التنفيذ:*** الخاصية الثانية هي زمن تنفيذ أسوأ حالة للخوارزمية المتوازية.

أ- t(n) يجب أن تكون صغيرة : الدافع الأساسي لبناء الخوارزمية المتوازيـة هـو تسريع العمليات الحسابية. ولذلك فإنه من المهم أن تكون الخوارزميات المتوازيـة التي نصممها أسرع بشكل مميز من أفضل خوارزمية تسلسلية للمسألة التي بـين أيدينا.

ب- t(n) يجب أن تكون متكيفة adaptive: فمثلاً قد يرغب شـخص مـا أن يملـك خوارزمية يتناقص زمن تنفيذها كلما كان عدد المعالجات المستعملة أكبـر. فمـن المرغوب أن t(n) تتغير بشكل متناسب (عكسياً) مع p(n) بدون مجموعة الحدود لـ p(n).

3. ***الكلفة :*** أخيراً نرغب بالحصول على خوارزميات متوازية يتحقق فيها :

$$C(n) = t(n) \times p(n)$$

والتي دائماً تناسب الحد الأدنى المعروف على عدد العمليات التسلسلية المطلوبة في أسـوأ حالـة لحل المسألة. وبمعنى آخر الخوارزمية المتوازية ستكون بأفضل كلفة.

إن اجتماع الموضوعات السابقة عادة يكون صعباً وفي بعض الأحيان مستحيلاً. وبشكل خاص عنـدما تكون المعالجات المتعددة موصولة عن طريق شبكة اتصالات حيث أن الشكل الهندسي للشبكة يفرض حدوداً على ما نستطيع إتمامه عن طريق الخوارزمية المتوازية. ولكن الأمر مختلف تماماً عندما تكون الخوارزمية منفذة على حاسب متوازي بذاكرة مقسمة، فمن غير المعقول هنا أن نصـر علـى هـذه الخواص المعطاة إذ أن النموذج يكون قوياً ومرناً.

الفصل الثالث

خوارزميات متوازية لحل بعض المسائل على أجهزة متعددة المعالجات

3-1 خوارزمية التحقق من الانتهاء من التوزيع (خوارزمية النشر):

لقد رأينا في الفصل الأول أنه في النموذج EREW SM SIMD لا يمكن الوصول المحدد إلى نفس موقع الذاكرة لمعالجين معاً بآن واحد ومع ذلك فقد يكون من الضروري أن تقرأ عدة معالجات عنصر بيانات محدد محفوظ في موقع معين من الذاكرة المشتركة.

بمعنى آخر فقد نحتاج أحياناً إلى نشر عنصر بيانات ما على جميع المعالجات في الحاسب المتوازي من أجل استعماله من قبلها جميعاً في الوقت نفسه.

ومن الواضح أن المحاكاة الفعالة لهذه العملية لا يمكن إنجازها في خطوة واحدة على نموذج EREW ونحتاج من أجل تلك المحاكاة إلى وضع إجراء ينفذ هذه المهمة. سنقوم أولاً بتوضيح مهمة هذا الإجراء على مثال.

مثال (3-1):

ليكن لدينا ملف كبير جداً يتألف من n إدخال منفصل ولنفترض أن الملف غير مرتب بأي طريقة. فإذا أردنا البحث عن عنصر معين مثل x في هذا الملف على حاسوب من النموذج EREW SM SIMD مع N معالج حيث $N < n$ ، عندئذ نحتاج إلى نشر هذا العنصر على جميع المعالجات ، لنرمز للمعالجات بـــ: $p_1,p_2,..,P_N$

تتم عملية النشر باستعمال الإجراء BROUDCASTING الذي يقوم بما يلي:

4- p_1 يقرأ x ويبلغ p_2

5- بنفس الوقت p_1,p_2 تبلغ p_3,p_4 على التوالي

6- بنفس الوقت p_1,p_2,p_3,p_4 تبلغ p_5,p_6,p_7,p_8 على التوالي

وهكذا..

تستمر هذه العملية حتى تحصل جميع المعالجات على x حيث يتضاعف عدد المعالجات في كل مرحلة مما يعني أن هذه العملية تتطلب log n خطوة. لنفترض أن جميع المعالجات تحتاج عنصر بيانات معين في لحظة ما خلال تنفيذ الخوارزمية وليكن هذا العنصر محجوز في الموقع D من الذاكرة. تتم محاكاة عملية القراءة المتعددة هذه على حاسب EREW عن طريق عملية النشر Broadcasting والتي نعرفها بالإجراء BROUDCAST. يفترض هذا الإجراء وجود مصفوفة A طولها N في

الذاكرة ، حيث تكون هذه المصفوفة فارغة في البداية ويستعملها الإجراء كمكان لنشر محتويات D إلى المعالجات يرمز لها بـــ A(i).

خوارزمية الإجراء BROUDCAST(D,N,A) (خوارزمية النشر):

الخطوة1: - المعالج P_1 :

1- اقرأ القيمة في D

2- خزنها في الذاكرة الخاصة بـــ P_1

3- اكتبها في A(1)

الخطوة2: - من (i=0) إلى (log N-1)

من $(j=2^i+1)$ إلى (2^{i+1})

- المعالج P_j :

1- اقرأ القيمة في $A(j-2^i)$

2- خزنها في الذاكرة الخاصة بـــ P_j

3- اكتبها في A(j).

```
Void BROADCAST(D,N,A)

{

step1:      processor P1

            (i)    reads the value in D
            (ii)   stores it in its own memory , and
            (iii)  write it in A(1);

Step2:      for (i=0 , i < log N-1 , i++)

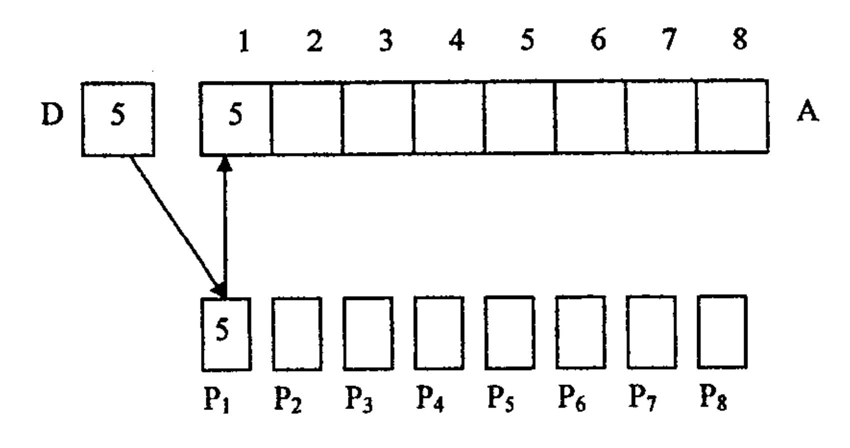
                for (j=2^i+1 , j < 2^(i+1), j++) in parallel

                    processor Pj

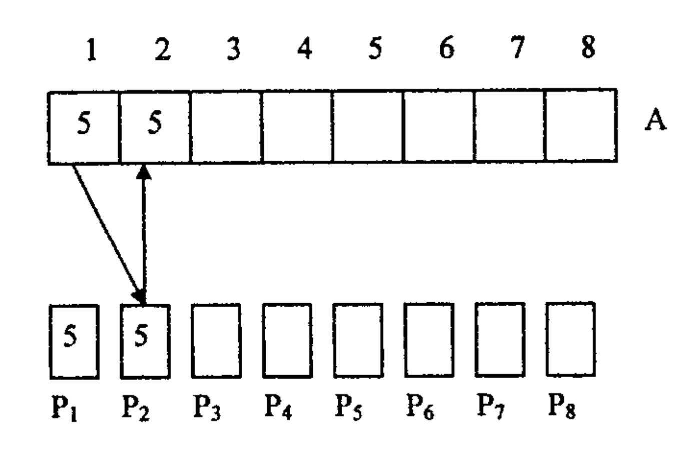
                    (i)    reads the value in A(j-2^i),
                    (ii)   stores it in its own memory , and
                    (iii)  writes it in A(j);

}
```

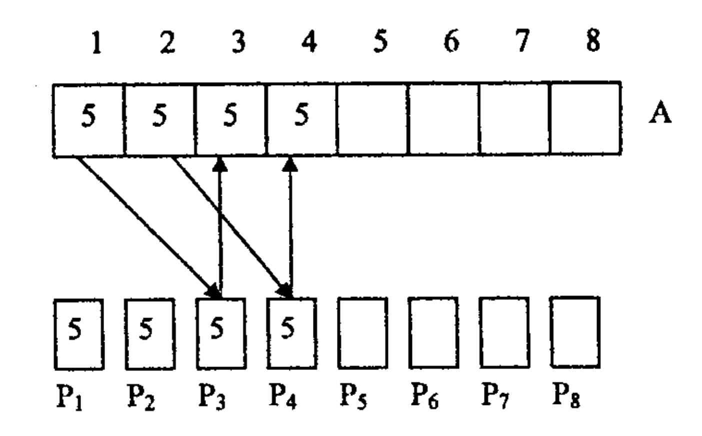
نوضح عمل الإجراء BROADCAST في الشكل (3-1) من أجل N=8 , D=5.



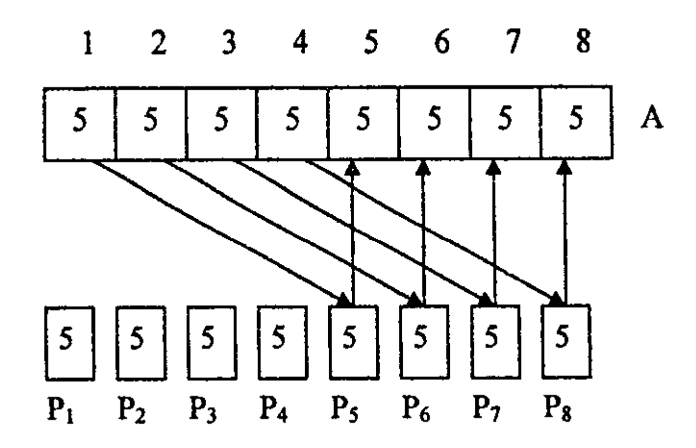
شكل (3-1 (أ)) تنفيذ الخطوة1



شكل (3-1 (ب)) تنفيذ الخطوة 2 عندما (i=0)



شكل (3-1 (جـ)) تنفيذ الخطوة 2 عندما (i=1)



شكل (3-1 (د)) تنفيذ الخطوة 2 عندما (i=2)

الشكل(3-1) شكل توضيحي لتنفيذ مهام الإجراء BROADCAST

عندما ينتهي الإجراء تكون جميع المعالجات قد خزنت قيمة D في ذاكرتها المحلية الخاصة من أجل استخدام آخر. طالما أن المعالجات المتعددة قد قرأت D مرتين في كل تكرار فإن الإجراء ينتهي في زمن $O(\log N)$.

إن الذاكرة المطلوبة من الإجراء BROADCAST هي مصفوفة طولها N وبكلام أدق مصفوفة من نصف ذلك الطول إذ انه في آخر تكرار للإجراء تكون جميع المعالجات قد استقبلت القيمة في D ولا تحتاج أن تكتبها مرة أخرى في A (انظر الشكل 3-1 (د)).

يمكن للإجراء BROADCAST بسهولة أن يتكيف لمنع الكتابة الأخيرة ومن أجل ذلك يستعمل مصفوفة A بطول N/2.

3-2 خوارزمية مراقبة الانتهاء:

عندما تعمل إحدى الخوارزميات على حاسب متعدد المعالجات ويقوم كل من هذه المعالجات بمهمة معينة للوصول إلى نتيجة ما عندئذ فإننا قد نحتاج أحياناً إلى معرفة ما إذا أنهت هذه المعالجات مهمتها أم لا ويتم ذلك كما يلي:

لنأخذ موقع f ، يحتوي هذا الموقع على قيمة منطقية تحدد فيما إذا أتم أحد المعالجات المهمة المطلوبة وبالتالي تنهي باقي المعالجات عملها. في بداية الأمر تأخذ f القيمة false وعندما ينهي المعالج مهمته

فإنه يضع في f القيمة true ، وفي كل خطوة تقوم جميع المعالجات بفحص قيمة f فإذا كانت true تتوقف عن العمل.

في حالة الحواسيب من النموذج EREW نلاحظ أن هذه العملية تحتاج إلى log N خطوة لنشر قيمة f في كل مرة تحتاج إليها المعالجات وهذا يزيد زمن التنفيذ. أما في نموذج الحواسـيب CREW فـإن عمليات القراءة المتعددة مسموحة وبالتالي تأخذ خطوة واحدة لجميع المعالجات من أجل الحصول على قيمة f مما يخفض من زمن التنفيذ. كما نلاحظ أنه ربما ينهي أحياناً أكثر من معالج مهمتــه فــي آن واحد فيحتاج عندئذ أكثر من معالج إبلاغ النجاح في الوقت نفسه وهذا يعني أن اثنــين أو أكثــر مــن المعالجات ستحاول الكتابة في الموقع f في الوقت ذاته وهذه العملية ممكنة فــي نمــوذج الحواســيب CRCW SM SIMD فقط.

3-3 خوارزمية سبر التفرع الشجري (خوارزمية البحث على الشجرة):

[GC.O,1993], [KMJm,1982] , [CSJJ,1981] يعتبر البحث أحد العمليات الشائعة في مجال الحسابات. ويستخدم في التطبيقات التي نحتاج فيها لمعرفة فيما إذا كان عنصر ما ينتمي إلى قائمة أو بشكل أعم تكرار في معلومات حقل مرتبطة بذلك العنصر.

تتوضح مسألة البحث بشكل قاعدة عامة كما يلي:

تعطى متسلسلة $S = \{s_1, s_2, \ldots\ldots\ldots s_n\}$ من الأعداد الصحيحة وعدد صحيح x والمطلوب تحديد فيما إذا كان $x = s_k$ حيث s_k عنصر من S. تحل هذه المسألة في الحسابات التسلسلية بمسح المتسلسلة S ومقارنة العنصر x مع عناصرها المحددة وتستمر هذه العملية حتى نجد عنصراً صحيحاً مساوياً لــx أو تعود العملية بالفشل.

لنفترض الآن أن لدينا حاسب SIMD باتصال شجري مع n مستوى من أجل البحث في ملف فيه n سجل. مثل الشجرة المبينة في الشكل من أجل $n = 16$.

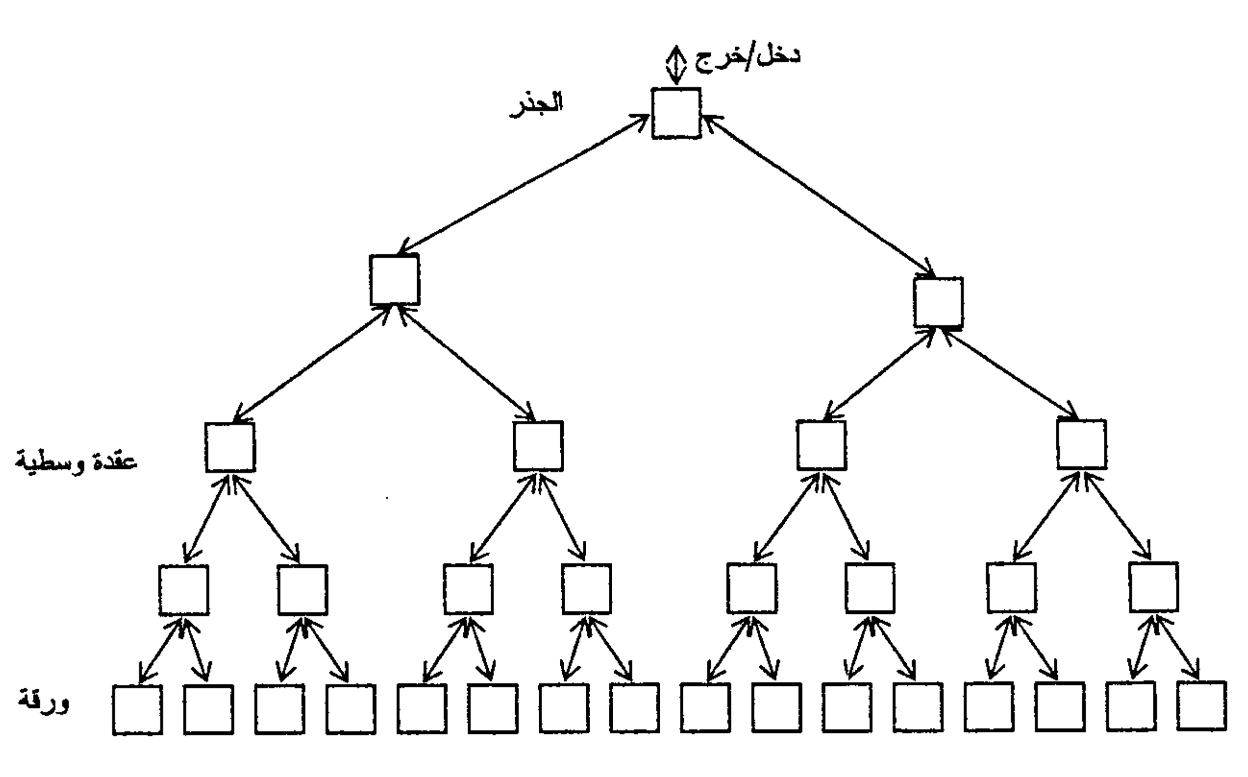
تختلف مهام المعالجات في هذا النموذج حسب توضعها في الشجرة :

- تخزن كل ورقة في الشجرة سجل واحد للملف الذي نبحث فيه.
- يتلقى الجذر الإدخال من العالم الخارجي ومن ثم يمرر ناتج الخرج الذي يتلقاه من أبنائه إلى العالم الخارجي.

- أما من أجل العقد الوسطى فكل منها يستطيع القيام بما يلي:

1. تتلقى العقدة الوسطى إدخالاً واحداً من الخلف (الأب) ويصنع منه نسختين يرسل واحدة إلى كل من أبنائه.

2. يتلقى إدخالين من ابنيه يقارنهما ويرسل النتيجة إلى والده.



/الشكل (3-2)/ البحث على حاسب باتصال شجري

سنوضح الآن عملية الاستعلام والحفظ للملف المخزن في الأوراق فيما يلي:

3-3-1 خوارزمية الاستعلام:

ليكن لدينا عدد صحيح x معطى والمطلوب البحث في سجلات ملف بالاعتماد على الحقل s من أجل x وتحديد فيما إذا كان هناك قيمة في $S = \{s_1, s_2, ... s_n\}$ مساوية لـ x ، يحتاج هذا الاستعلام فقط جواباً نعم أو لا وهذا شكل القاعدة العامة للاستعلام.

ينفذ الاستعلام في حاسب الاتصال الشجري بمروره في ثلاث مراحل:

المرحلة1: يقرأ الجذر x ويمررها إلى ولديه. وبدورها هذه ترسل x إلى أولادها ، تستمر العملية حتى تصل نسخة من x إلى كل ورقة.

المرحلة2: تقارن كل الأوراق بآن واحد الحقل s للسجل المخزن فيها مع x: إذا كان مساوياً فإن الورقة تنتج 1 كخرج و إلا فإنها تنتج 0 .

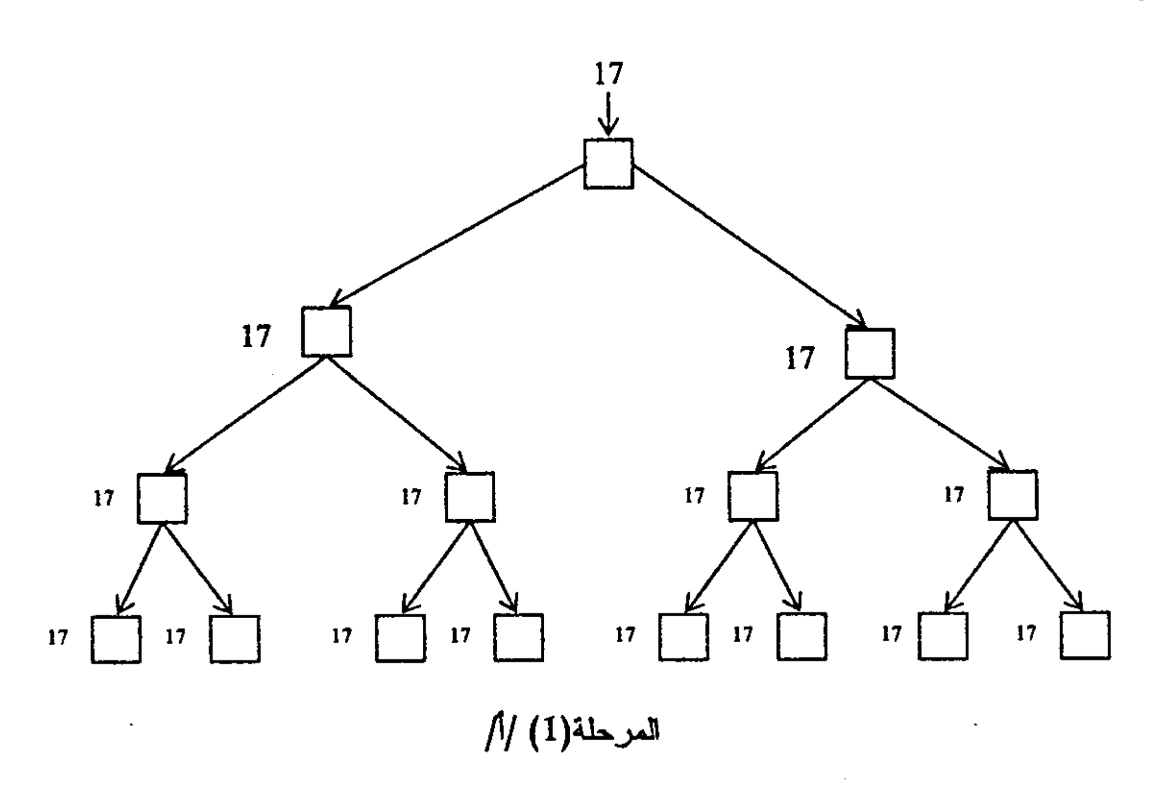
المرحلة3: يندمج خرج الأوراق بشكل متقدم في الشجرة: كل عقدة متوسطة تحسب العامل OR المنطقي لإدخاليها.

(مثال: 0 = 0 or 0 , 1 = 0 or 1 , 1 = 1 or 1) وتمرر النتيجة إلى والدها. تستمر العملية حتى يتلقى الجذر خانتين تحسب لهما العامل or المنطقي وينتج إما 1 للإيجاب أو 0 للنفي.

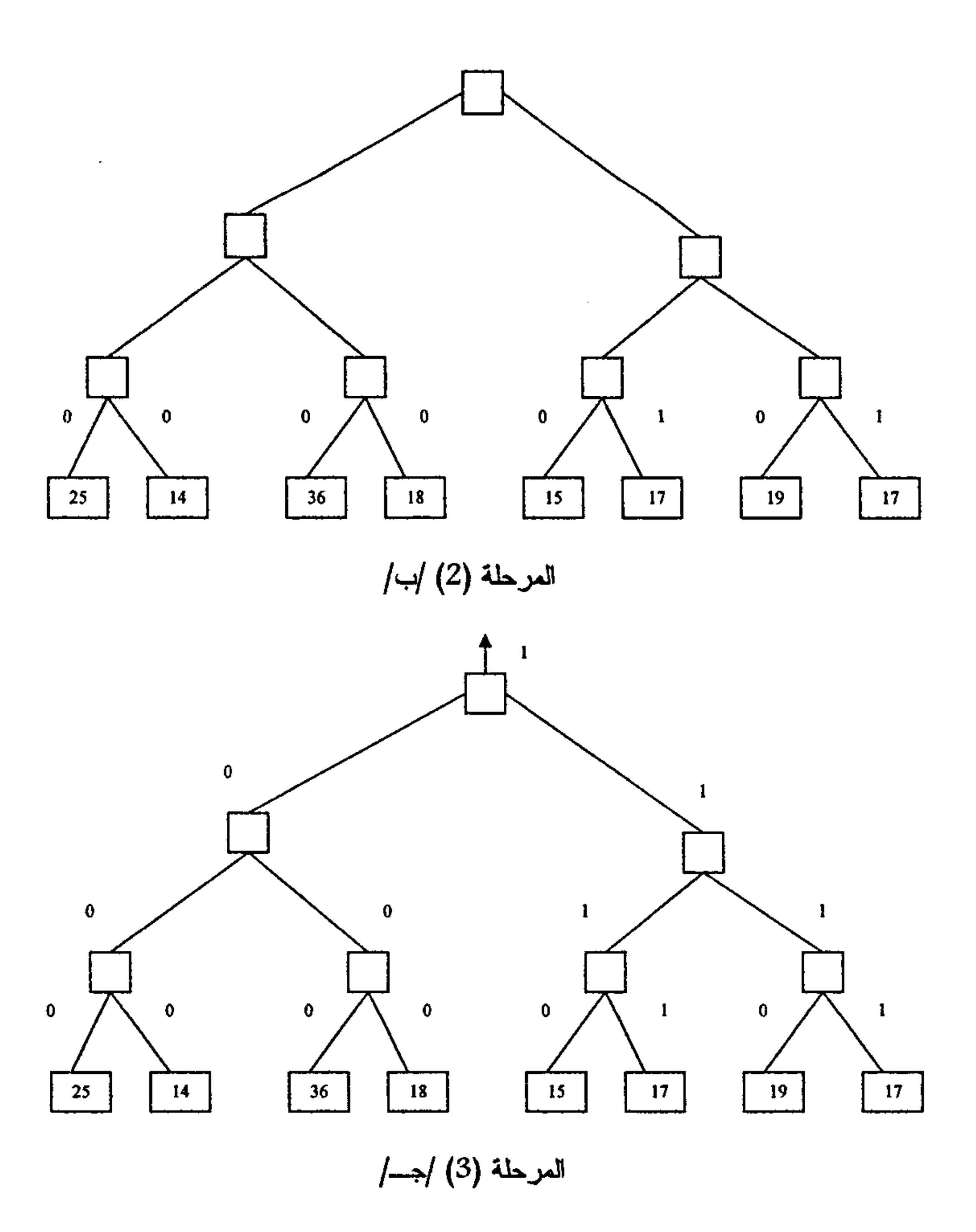
نلاحظ أن مثل هذا الاستعلام تتم الإجابة عنه في زمن $O(\log n)$.

مثال (3-2):

لتكن {25,14,36,18,15,17,19,17} = S و 17 = x. سنقوم بتوضيح المراحل الثلاثة فيما يلي في الشكل:



المرحلة(1) /أ/



المرحلة (2) /ب/

المرحلة (3) /جـ/

شكل(3-3) /البحث في متسلسلة من 8عناصر باستعمال الشجرة/

لنفرض الآن أن عدداً من هذه الاستعلامات وليكن q يصطف بانتظار المعالجة. ويمكن أن تمرر مباشرة إلى أسفل الشجرة طالما أن الجذر والعقد الوسطى شاغرة لمعالجة الاستعلام التالي حالما تقوم بتمرير الاستعلام الحالي إلى الأمام لأبنائهم. بالمقابل نجد أنه بالنسبة للأوراق حالما تنتج نتيجة أحد المقارنات فإن الورقة تصبح جاهزة لتلقي قيمة جديدة لـــ x وتمرر النتائج أيضاً مباشرة للأعلى.

يستطيع الجذر والعقد الوسطى حساب العامل or المنطقي للزوج التالي من البتات حالما ينتهي من الزوج الحالي ، وبالتالي نجد أن الجذر والعقد الوسطى تتلقى تدفق البيانات بشكل منحدر (للاستعلامات) وبشكل صاعد (للنتائج) بآن واحد ، وقد افترضنا أن كليهما يمكن أن يعالج في وحدة زمنية مفردة وعلى الرغم من حفظ كليهما لتدفقات حركة البيانات ، فإن المعالج يمكنه أن يعمل بالتناوب من أحد الاتجاهات إلى الآخر. مما يعني أن عملية الإجابة عن الاستعلام الأول الذي سينتج في الجذر تأخذ زمناً قدره: $O(\log n)$ ، وبالتالي فإننا نحصل على الإجابة عن الاستعلام الثاني من وحدة الزمن التالية، وهكذا فإننا نحصل على الإجابة عن الاستعلام الأخير بعد $(q-1)$ وحدة زمن من الإجابة الأولى.

أي أن نتيجة الـــ (q) إجابة يمكن أن نحصل عليها بكلفة زمنية هي: $O(\log n) + O(q)$.

سنقوم الآن ببعض التغييرات على القاعدة من أجل الاستعلام المناقش حتى الآن:

1.خوارزمية الموضع:

إذا نجح الاستعلام وكان العنصر s_k مساوياً لـــ x فإن المطلوب أن نعرف الدليل k. لنفترض أن الأوراق مرقمة من 1 إلى n وأن الورقة i تحوي s_i. وبعد مقارنة المعالجات الأوراق للبيانات مع x نجد أنه إذا كان $s_i=n$ تنتج الورقة i الزوج $(1,i)$ و إلا فإنها تنتج (o,i).

أما بالنسبة للعقد الوسطى والجذر فهي تعمل كما يلي :

إذا وصل الزوج $(1,i)$ و$(0,j)$ عندها يُرسل الزوج $(1,i)$ للأعلى. و إلا إذا كان كلا الزوجين يحويان 1 كعنصر أول أو إذا كان كلا الزوجين يحويان 0 كعنصر أول عندها يُرسل الزوج الذي وصل من الابن الأيسر للأعلى ، وبهذه الطريقة ينتج الجذر إما:

1. $(1,k)$ حيث k هو الدليل الأصغر لعناصر S المساوية لـــn ، أو
2. $(0,k)$ مما يعني أنه لا يوجد أي عنصر مساوٍ لـــx ولذلك فإن قيمة k لا معنى لها.

مع هذه التعديلات فإن الجذر في المثال السابق ينتج $(1,6)$.

يمكن تنفيذ هذا التغيير في قاعدة الاستعلام في ثلاث حالات:

أ- إذا كان المطلوب هو الحصول على كل معلومات السجل كجواب للاستعلام (أو ربما بعض حقوله) عندئذ إذ وُجِد السجل الذي يتحقق فيه أن الحقل s مساوي لـــx ، فيمكن أن يأخذ ناتج الورقة

ثلاثية من الشكل (المعلومات المطلوبة,1,i). أما بالنسبة للجذر والعقد الوسطى فإنها تتصرف كما سبق.

ب- إذا كان المطلوب هو الحصول على جميع مواقع العناصر المساوية لـــ x في S ، في هذه الحالة عندما يتلقى الجذر أو العقدة الوسطى زوجين (1,i) و (1,j) فإن الزوجين يرسلا إلى الأعلى بشكل متتابع (تسلسلي) وبهذه الطريقة تنتج أدلة جميع عناصر S المساوية لـــ x في النهاية من الجذر.

ت- الحالة الثالثة هي جمع بين (أ) و(ب): فإذا كان المطلوب هو الحصول على جميع معلومات السجلات التي فيها الحقل s مماثل لـــ x. وتعالج هذه الحالة عن طريق الجمع بين الحالتين السابقين.

يجب الأخذ بعين الاعتبار في الحالات السابقة تزامن وجود عدة استعلامات على الخط. وهذا بسبب إرسال النتيجة للأعلى عن طريق كل عقدة وهي ليست أقل من خانة مفردة ولكن تقريباً عدة خانات من المعلومات ربما عدة سجلات (ففي أسوأ الأحوال تتألف الإجابة من الـــ n سجل المدخل) ، وبما أن الإجابة على الاستعلام لا يمكن التنبؤ بطولها فإنه بالتأكيد لا يوجد أطول من أسوأ حالة للاستعلام الذي يأخذ كلفة زمنية $O(\log n)$ وهذه المدة ثابتة وفي حالة q استعلام نجد أنه ينفذ في زمن قدره $O(\log n) + O(q)$.

2. خوارزمية العدد:

قد نحتاج في بعض الأحيان إلى تغيير قاعدة الاستعلام فنسأل عن عدد السجلات التي يتساوى فيها الحقل s مع x . وتتم معالجة هذه الحالة تماماً مثل الاستعلام الأساسي ما عدا أنه الآن نقوم بحساب مجموع إدخالات العقد الوسطى والجذر (إضافة إلى العامل or المنطقي). ومع هذا التعديل يُنتج الجذر في المثال السابق العدد 2.

3. خوارزمية أقرب عنصر:

ربما يكون من المفيد أحياناً إيجاد العنصر من S بحيث تكون قيمته هي الأقرب لـــ x . تتم المعالجة هنا كما في الاستعلام الأساسي ، حيث تكون x أول إرسال للأوراق ، ومن ثم تحسب الورقة i القيمة المطلقة لـــ s_i-x وتدعوها a_i وتنتج (i,a_i) كخرج. يتلقى الآن كل من العقد الوسطى والجذر زوجين (i,a_i) و (j,a_j): ومن ثم يرسل الزوج الذي يحوي a الأصغر للأعلى.

في المثال السابق مع هذا التعديل إذا كان الدخل x=38 فإن الجذر سينتج (3,2) كخرج . تعالج حالة وجود زوجين يحويان المكون a نفسه إما باختيار أحدهما عشوائياً أو بإرسال الاثنين تسلسلياً.

4. خوارزمية الرتبة:

رتبة عنصر x في S تعرف كما يلي: هي (عدد العناصر الأصغر من x) + 1 في S. نعالج هذه الحالة كالتالي: نبدأ بإرسال x إلى الأوراق ومن ثم نجعل كل ورقة i تنتج 1 إذا كان $x > s_i$ و0 فيما عدا ذلك.

نحسب الآن رتبة x في S بجعل جميع العقد الوسطى تضيف إدخالاتها وترسل النتيجة للأعلى. يضيف الجذر واحد إلى مجموع إدخاليه قبل إنتاجه كرتبة. في المثال السابق يكون خرج الجذر مع هذا التعديل هو 3. ونلاحظ أن كل التعديلات السابقة تأخذ نفس زمن التنفيذ كما في الاستعلام الأساسي.

3-4 تطبيقات الخوارزميات المتوازية:

سنعرض الآن بعض تطبيقات مسائل القرار والأمثلية التي نستخدم فيها خوارزميات متوازية فعالة مثل خوارزمية حساب المجاميع البادئة لسلسلة وقد اخترنا مسألتين هما:

مسألة الأعمال المتسلسلة وفق فترات انتهائها (job sequencing with dead lines) وهي مسألة قرار و مسألة حقيبة الظهر (knapsack problem) وهي مسألة حل أمثل.

سنأخذ في كل من المسألتين خوارزمية تعمل على حاسب متوازي باتصال شجري ، علماً أن الخطوة الأساسية في كل من الخوارزميتين هي حساب المجاميع البادئة لسلسلة، وسنبدأ أولاً بتوضيح عملية حساب المجاميع البادئة (prefix sums).

3-4-1 حساب المجاميع البادئة:

لتكن لدينا سلسلة مؤلفة من n عدد $X=\{x_0,x_1,x_2,..,x_{n-1}\}$ حيث $n\geq1$. ولنفرض أن n هو قوة لـــ 2 ويمكننا في غير هذه الحالة إضافة عناصر مساوية للصفر إلى السلسلة حتى يصبح حجمها قوة لـــ 2 ، ولنحسب جميع المجاميع البادئة للسلسلة X ولتكن: $S = \{s_0,s_1,..,s_{n-1}\}$

حيث: $s_i = x_0+x_1+x_2+..+x_{n-1}$ ، من أجل: $i=0,1,..,n-1$.

أما إذا كانت عناصر X تشكل سلسلة كالتالي: $X= x_0x_1x_2..x_{n-1}$ عندها تكون s_i هي مجموع العناصر المشكلة لبادئة بالطول i.

يمكن حساب المجاميع الـ n البادئة تسلسلياً في زمن O(n) بالإجراء التالي:

1- خوارزمية الإجراء : SEQUENTIAL SUMS(X,S)

الخطوة 1: $s_0 \leftarrow x_0$

الخطوة 2: من أجل i=1 إلى n-1 نفذ

$s_i \leftarrow s_{i-1} + x_i$

```
Void SEQUENTIAL SUMS(X,S)
{
step1:      s0 ← x0 ;
step2:      for (i=1 , i < n , i++)
                  si ← si-1 + xi ;
}
```

ويعتبر زمن التنفيذ هذا أمثلياً إذ أننا نحتاج إلى Ω(n) خطوة لقراءة الدخل.

وعلى العكس عندما يتوفر عدة معالجات كل منها قادر على إنجاز عملية إضافة فإنه يمكننا الحصول على السلسلة $S=\{s_0,s_1,..,s_{n-1}\}$ بشكل أسرع. ونبين ذلك بالإجراء PARALLEL SUMS فيما يلي:

2- خوارزمية الإجراء : PARALLEL SUMS(X,S)

الخطوة 1: من أجل i=0 إلى n-1 نفذ على التوازي

$s_i \leftarrow x_i$

الخطوة 2: من أجل j=0 إلى ((log n)-1) نفذ

من أجل $i=2^j$ إلى n-1 نفذ على التوازي

$s_i \leftarrow s_{i-2^j} + s_i$

```
Void PARALLEL SUMS(X,S)
{
step1:      for (i=0 , i < n , i++) in parallel
                  si ← xi ;
step2:      for (j=0 , j < log n , j++)
                  for (i=2^j , i < n , i++) in parallel
```

$s_i \leftarrow s_{i-2^j} + s_i$;

}

يعتمد هذا الإجراء على المضاعفة التعاودية ويمكن لهذا الإجراء أن يطور على حاسب SIMD بذاكرة مشتركة وذلك باستعمال الإجراء ALLSUMS الذي يحتاج إلى n معالج $p_0,p_1,..,p_{n-1}$ وتأخذ المعالجات p_i قيماً ابتدائية x_i وتعطي نتائج نهائية s_i. ينفذ الإجراء في زمن O(log n) من أجل كلفة O(n log n) وتعتبر هذه الكلفة ليست أمثلية إذ أن مسألة حساب المجاميع البادئة تحتاج O(n) عملية تسلسلية بشكل كافٍ، وسنبين ذلك فيما يلي:

3-4-2 حساب جميع المجاميع Computing All Sums:

لنفترض أن لدينا عدد N من المعالجات كل معالج p_i يحجز في ذاكرته المحلية عدد a_i ؛ $1 \le i \le N$.

والمطلوب حساب المجموع $a_1+a_2+..+a_i$ من أجل كل معالج p_i (المعروفة بالمجاميع البادئة the prefix sums).

والسؤال هو: هل يمكن لنموذج الحاسبات SIMD بذاكرة مشتركة المتوفر من أجل حساب جميع المجاميع من الشكل $a_1+a_2+..+a_i$؛ $1 \le i \le N$ استعمال N معالج في زمن O(log N)؟

هذا محتمل فعلاً، فالفكرة هي جعل جميع المعالجات تعمل بآن واحد ما دامت هناك قدرة على استمرار عملية الإضافة. ويمكن أن تعطى خوارزمية الإجراء ALLSUMS كالتالي:

خوارزمية الإجراء $ALLSUMS(a_1,a_2,..,a_N)$:

من أجل j=0 إلى log N-1 نفذ

من أجل $i=2^j+1$ إلى N نفذ على التوازي

المعالج P_i:

1- احصل على القيمة a_{i-2^j} من المعالج p_{i-2^j} من خلال الذاكرة المشتركة

2- أبدل a_i بـ $a_{i-2^j}+a_i$

```
Void ALLSUMS(a1,a2,..,aN)

{

for (j=0 , j <= log N-1 , j++)

        for ( i=2^j+1 , i <=N , i++)

                processor Pi

                (i)    obtains a(i-2^j) from p(i-2^j) through shared memory , and
                (ii)   replaces ai with a(i-2^j)+ai ;

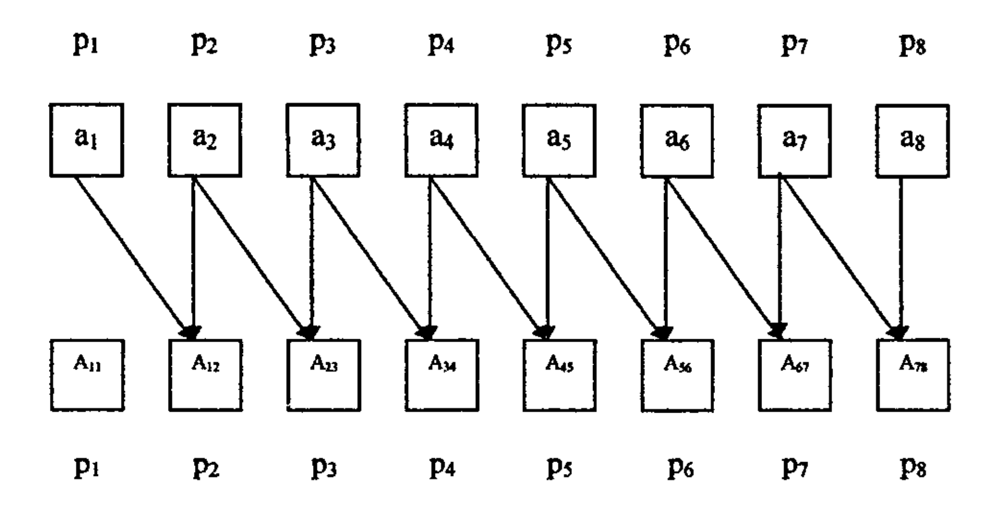
}
```

نوضح عمل الإجراء ALLSUMS في الشكل (3-4) من أجل N=8 ونرمز بـ A_{ij} إلى المجموع $a_1+a_2+..+a_i$.

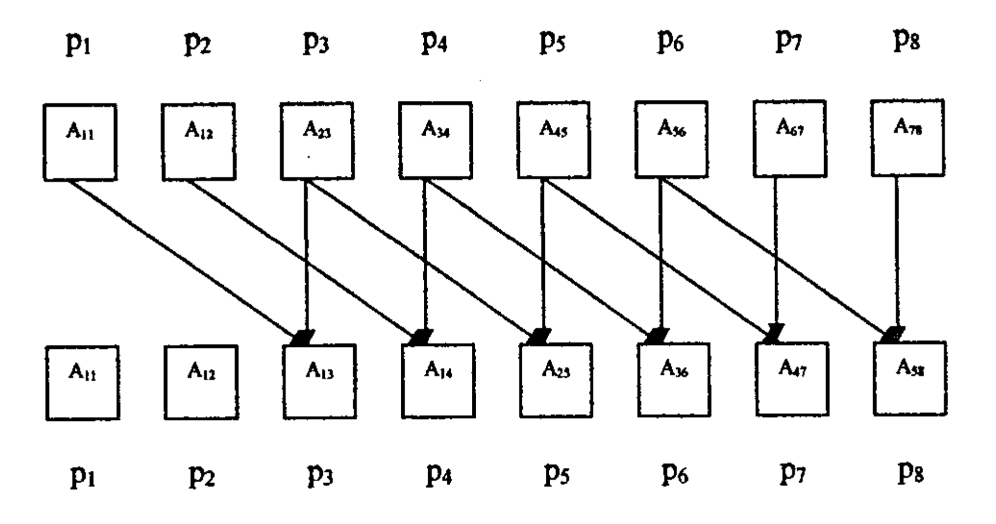
في بداية الإجراء تبدل a_i بـ $a_1+a_2+..+a_i$ في الذاكرة المحلية لـ p_i من أجل $1 \leq i \leq N$.

يتطلب الإجراء زمن O(log N) وذلك لأن عدد المعالجات التي انتهت من حسابها يتضاعف في كل مرحلة.

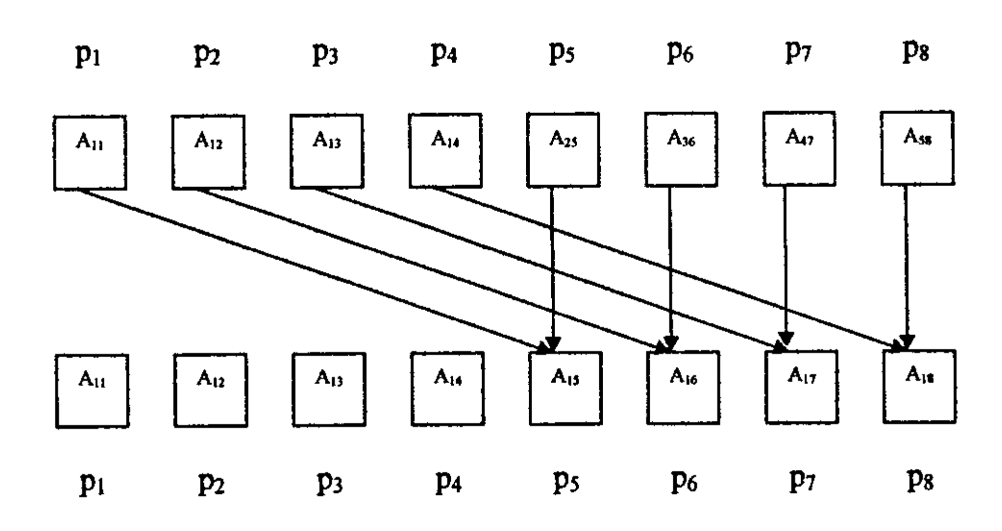
إن الإجراء ALLSUMS يمكن أن يكون مكيفاً لحل أي مسألة حيث يمكن إبدال عملية الإضافة بأي عملية ثنائية ترابطية أخرى مثلاً: ضرب عددين ، إيجاد أكبر أو أصغر العددين، وهكذا.. وهناك عمليات أخرى تطبق على زوج من الكميات المنطقية (أو أزواج من البتات) هي and , or, xor.



الشكل (3-4 / أ) من أجل القيمة j=1



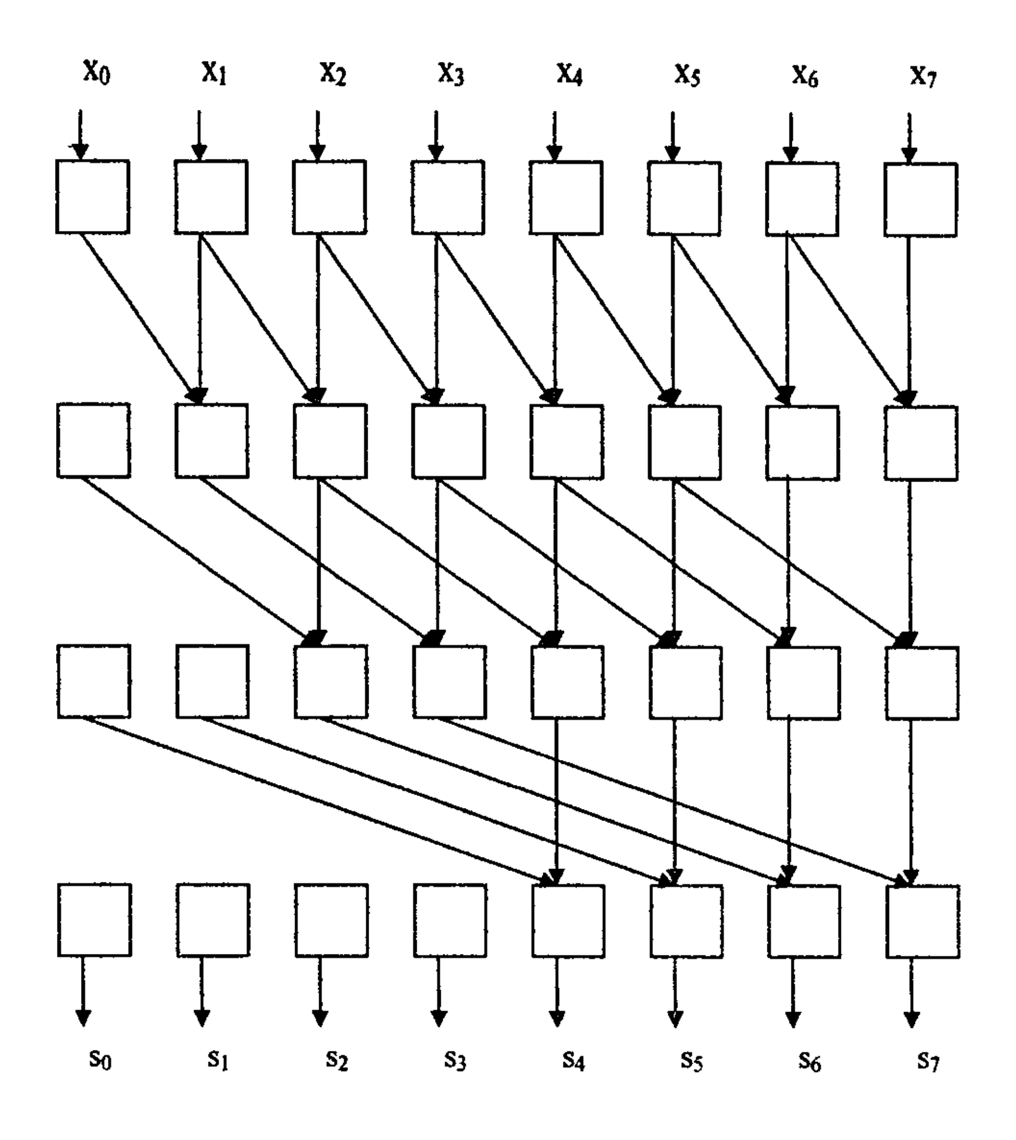
الشكل (3-4 / ب) من أجل القيمة j=2



الشكل (3-4/ جـ) من أجل القيمة j=3

الشكل (3-4) يوضح عمل الإجراء ALLSUMS من أجل N=8 حيث $A_{ij} = a_1+a_2+..+a_j$

بدمج هذه الأشكال الثلاثة نحصل على شبكة خاصة صممت خصيصاً من أجل تنفيذ الإجراء PARALLEL SUMS على نموذج الحسابات SIMD بذاكرة مشتركة.



شكل (3-5) شبكة تطوير خاصة للمضاعفة التعاودية

نلاحظ من الشكل (3-5) أن هذه الشبكة الخاصة تحوي 1+log n سطر كل منها يحوي n معالج وبالتالي يكون عدد المعالجات في الشبكة n + n log n.

لنفترض أن المعالجات في كل سطر مرقمة من 0 إلى n-1 وأن الأسطر مرقمة من 0 إلى log n.

يتلقى المعالج i في السطر j+1 الدخل من:

1- المعالج i في السطر j ،

2- المعالج $i-2^j$ في السطر j إذا كان $i \geq 2^j$

ويقوم كل معالج بحساب مجموع إدخاليه ويرسل النتيجة إلى السطر التالي من المعالجات المستعملة في الشبكة. يتلقى كل معالج دخلاً واحداً فقط ويمرر ذلك الدخل إلى السطر التالي، وهكذا تدخل عناصر

السلسلة إلى الشبكة من نهاية واحدة (عنصر واحد لكل معالج) ويتلقى الخرج من النهاية الأخرى. وتحسب جميع المجاميع البادئة بزمن قدره $O(\log n)$ ويعتبر هذا أفضل احتمال لزمن التنفيذ حيث أن الحد الأدنى هو $\Omega(\log n)$ وكلفة الشبكة $O(n \log^2 n)$، ومنه نجد أن هناك نموذج حسابات أضعف من نموذج الذاكرة المشتركة قادر على الوصول إلى زمن تنفيذ كهذا مثل الإجراء ALLSUMS الذي يستعمل عدداً أكبر من المعالجات.

3-4-3 خوارزمية المجاميع البادئة على شجرة:

سنشرح الآن خوارزمية متوازية لحساب المجاميع البادئة تعمل على شجرة ثنائية من المعالجات التي تعمل بشكل متزامن ، ولتكن إدخالات الخوارزمية هي عناصر السلسلة $x_0,x_1,..,x_{n-1}$ كل منها لواحد من المعالجات الأوراق الــn من الشجرة الثنائية $p_0,p_1,..,p_{n-1}$ ، وعندما تنتهي الخوارزمية فإن كل معالج p_i ينتج s_i ، وخلال الخوارزمية تنفذ المعالجات الأوراق والمعالجات الوسطى والمعالج الجذر عمليات بسيطة سنشرحها من أجل كل نوع من المعالجات:

أولاً خوارزمية المعالج الجذر:

1- إذا وصل الإدخال من الولد الأيسر عندها أرسله على الولد الأيمن.
2- إذا وصل الإدخال من الولد الأيمن احذفه.

ثانياً خوارزمية المعالجات الوسطى:

1- إذا وصل الإدخال من الولد الأيسر والولد الأيمن عندها:
- أرسل مجموع الإدخالين إلى الأب.
- أرسل الإدخال الأيسر إلى الولد الأيمن.

2- إذا وصل الإدخال من الأب عندها : أرسله إلى الأولاد الأيمن والأيسر.

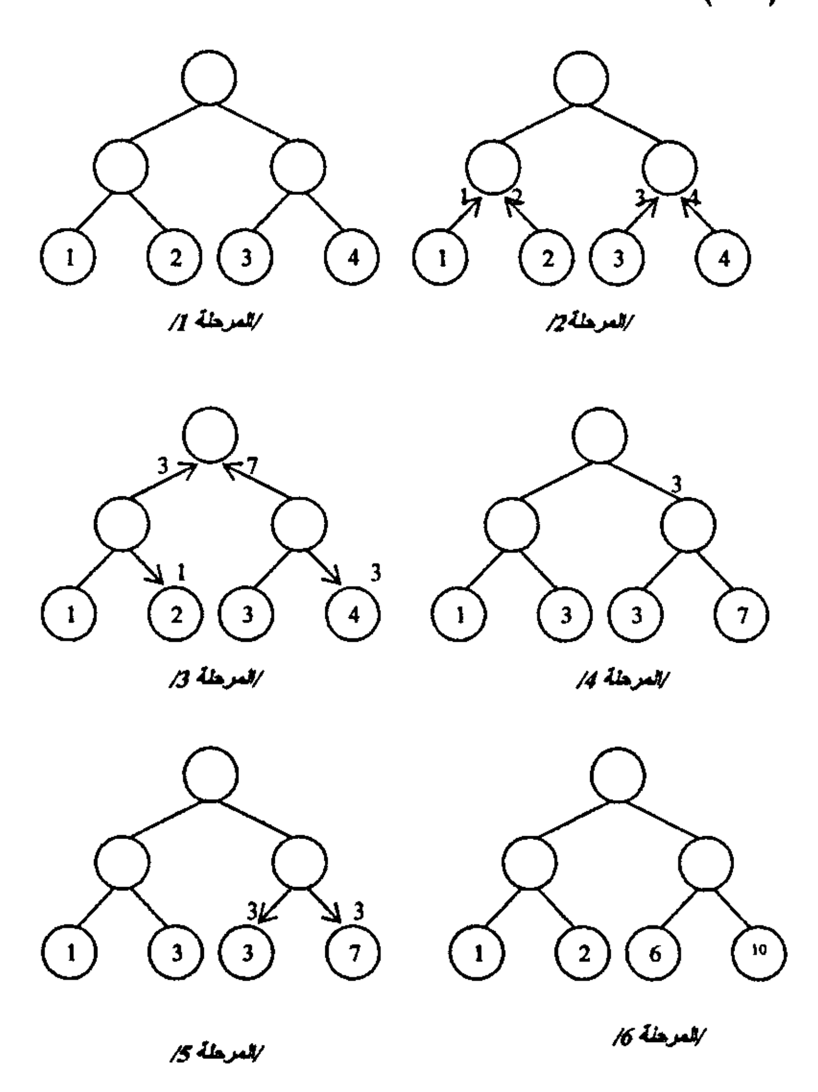
ثالثاً خوارزمية المعالجات الأوراق:

1- ضع القيم x_i في s_i.
2- أرسل قيمة x_i على الأب.
3- إذا وصل الإدخال من الأب عندها أضفه إلى s_i.

نلاحظ أن المعالجات الوسطى والمعالج الجذر تكون متأهبة للعمل عندما تتلقى إدخال، وبشكل مشابه بعد الحصول على x_i وإرسال s_i إلى الأب فإن المعالج الورقة يكون متأهباً أيضاً للعمل من خلال الإدخال الذي يتلقاه من والده. وبعد أن يتلقى المعالج الورقة الذي يقع في أقصى اليمين الإدخال رقم log n فإن القيم $s_0,s_1,..,s_{n-1}$ تصبح هي المجاميع البادئة لـ $x_0,x_1,..,x_{n-1}$.

مثال(3-3):

سنشرح الآن خوارزمية المجاميع البادئة على شجرة من خلال تطبيقها على السلسلة X={1,2,3,4} وذلك كما في الشكل (3-6):



الشكل (3-6) حساب المجاميع البادئة على شجرة المعالجات

3-4-4 تحليل خوارزمية المجاميع البادئة على شجرة:

إن عدد الخطوات المطلوبة من الخوارزمية هو البعد بين المعالج الورقة في أقصى اليمين والمعالج الورقة في أقصى اليسار ، وهذا البعد هو $(2 \log n)$ ولذلك فإن: $t(n) = O(\log n)$ ، وبما أن: $P(n) = 2n-1$ فإن: $C(n) = O(n \log n)$ وهذه الكلفة ليست أمثلية.

ويمكننا الحصول على خوارزمية لها كلفة أمثلية بسهولة وذلك من خلال زيادة فعالية المعالجات الأوراق أي زيادة المهام التي تنفذها. لتكن لدينا شجرة المعالجات بـــ N ورقة ولتكن $p_0,p_1,..,p_{N-1}$ حيث $n \geq N$ وسنفترض للسهولة أن n هو مضاعف للعدد N مع العلم أن الخوارزمية يمكن أن تتكيف للعمل مع جميع قيم n.

لنأخذ سلسلة الدخل $X = \{x_0,x_1,..,x_{n-1}\}$ في البداية يحتوي المعالج الورقة p_i على العناصرالتالية: $x_{i(n/N)}$, $x_{i(n/N)+1}$, .. , $x_{i(n/N)+(n/N)-1}$. بالنسبة للمعالج الجذر والمعالجات الوسطى تنفذ الأعمال نفسها التي تم شرحها مسبقاً وسنناقش الآن عمل المعالجات الأوراق التي تنفذ الخطوات المعطاة في الإجراء التالي. سنرمز فيما يلي بـــ v_i إلى عدد يتألف من خانة واحدة في التمثيل الثنائي للعدد i أي أن:

$$v_o = 0$$
$$v_i = 1 + v_{i \bmod 2^{(\log i)}}$$

حيث : $m = n/N$

1- خوارزمية المعالج الورقة:

1- احسب جميع المجاميع البادئة لـــ x_{im} , x_{im+1} , ... , x_{im+m-1} ،
وخزن النتائج في: s_{im},s_{im+1} , ... , s_{im+m-1} ،
وأرسل s_{im+m-1} إلى المعالج الأب.

2- ضع المجموع المرحلي (المؤقت) r_i صفراً.

3- إذا كان الدخل قد وصل من الأب عندها أضفه إلى r_i .

4- إذا كان r_i هو مجموع v_i دخل متلقى من الأب تماماً عندها أضف r_i إلى كل من التالي:
s_{im} , s_{im+1} , ... , s_{im+m-1}

لكي نفهم شرط الانتهاء في المرحلة (4) نلاحظ أن v_i هي عدد جذور الأشجار الفرعية على يسار p_i تماماً وهي التي ترسل الإدخال إلى p_i .

2- تحليل خوارزمية المعالج الورقة:

يعتمد عدد البيانات المطلوبة من الخوارزمية للتجوال أعلى وأسفل الشجرة على عدد العناصر المخزنة في كل معالج ورقة. مما يؤدي إلى أن زمن تنفيذ الخوارزمية هو مجموع ما يلي:

1- الزمن المطلوب من الورقة p_i لحساب $s_{im}, s_{im+1}, \ldots, s_{im+m-1}$ ، ومن ثم إرسال s_{im+m-1} إلى الأب (أي أن الزمن هو O(n/N)) طالما أن جميع الأوراق تنفذ هذه الخطوة بآن واحد.

2- الزمن المطلوب من الورقة التي في أقصى اليمين p_{N-1} لتلقي إدخالها النهائي (أي إلى زمن O(log N)).

3- الزمن المطلوب من الورقة في أقصى اليمين p_{N-1} (وهو آخر معالج يتوقف) لإضافة r_{N-1} إلى كل المجاميع التي يحتويها (أي أن الزمن هو O(n/N)).

ومنه : $t(n)= O(n/N) + O(\log N)$

وبما أن : $p(n) = 2N-1$ و $c(n) = O(n + N \log N)$ فهذا يعني أن الخوارزمية يكون لها كلفة أمثلية إذا كان $N \log N = O(n)$.وعلى سبيل المثال إن $N=O(n/\log n)$ ستنجح بالوصول إلى كلفة أمثلية.

يجب أن نلاحظ أن أمثلية كلفة الخوارزمية تعود أصلاً إلى حقيقة أن الزمن المطلوب في الحسابات داخل الأوراق يتحكم في الزمن المطلوب من المعالجات للاتصال فيما بينها. وهذا قد أوصلنا إلى تجزئة مسألة المجاميع البادئة إلى مسائل جزئية مجزأة وهذا يتطلب كمية صغيرة من الاتصالات فقط.

3-4-5 المجاميع البادئة على الشبكة Mesh:

نبين فيما يلي كيفية حساب المجاميع البادئة لسلسلة على مصفوفة معالجات متصلة شبكياً. تعود رغبتنا في دراسة الخوارزمية المتوازية لحل المسألة على هذا النموذج لسببين:

عندما يكون الزمن المأخوذ من تجوال وحيد على طول السلك متناسباً مع طول ذلك السلك تكون الشبكة هي الأفضل من الشجرة لحل عدد من المسائل وهذه المسائل تتصف بأن زمن حلولها تتناسب مع المسافة:

1- من الجذر إلى الورقة في الشجرة

2- من السطر العلوي إلى السطر السفلي في الشبكة.

وتعتبر مسألة حساب المجاميع البادئة لسلسلة مدخلة هي واحدة من هذه المسائل.

كما أنه يمكن لشبكة بـ n معالج أن تخزن سلسلة بـ n إدخال بشكل أسرع من شجرة بـn ورقة بغض النظر عن أية افتراضات قمنا بها حول زمن النشر الوحيد على طول الأسلاك. وطالما أن الترتيب عنصر هام في حلنا للمسائل الموصوفة سنفرض لسهولة التعريف فيما يلي أن n هو مربع كامل ولتكن $m = n^{1/2}$. يمكن حساب المجاميع البادئة لـ $X=\{x_0,x_1,..,x_{n-1}\}$ على حاسب باتصال شبكي m×m كما يلي: ولتكن المعالجات هي $p_0,p_1,..,p_{n-1}$ مرتبة سطرياً بترتيب أعظمي. في البداية تحتوي p_i على x_i وعندما تنتهي الخوارزمية يحتوي p_i على s_i.

تتألف الخوارزمية من ثلاث خطوات:

في الخطوة الأولى: يتم العمل في جميع الأسطر على التوازي حيث تحسب المجاميع البادئة للعناصر في كل سطر بشكل تسلسلي: فيضيف كل معالج على محتوياته محتوى جاره الأيسر.

في الخطوة الثانية: تحسب المجاميع البادئة للمحتويات في العمود اليميني. أخيراً مرة أخرى تتم جميع عمليات الأسطر على التوازي حيث تُضاف محتويات المعالج اليميني في السطر k-1 إلى جميع المعالجات في السطر k (ما عدا اليميني).

تعطى الخوارزمية فيما يلي كإجراء MESH PREFIX SUMS نرمز فيه إلى محتويات المعالج في السطر k والعمود j بـ u_{kj} حيث: $0 \le j \le m-1$ و $0 \le k \le m-1$.

خوارزمية الإجراء MESH PREFIX SUMS(X,S):

الخطوة 1: من أجل k=0 إلى m-1 نفذ على التوازي

من أجل j=1 إلى m-1 نفذ

$u_{kj} \leftarrow u_{kj} + u_{k,j-1}$

الخطوة 2: من أجل k=1 إلى m-1 نفذ

$u_{k,m-1} \leftarrow u_{k,m-1} + u_{k-1,m-1}$

الخطوة 3: من أجل k=1 إلى m-1 نفذ على التوازي

من أجل j=m-2 إلى 0 نفذ

$u_{kj} \leftarrow u_{kj} + u_{k-1,m-1}$

```
Void MESH PREFIX SUMS(X,S)

{

step1:      for (k=0 , k < m , k++) in parallel
```

```
            for (j=1 , j < m , j++)
                    u_kj ← u_kj + u_k,j-1 ;
step2:      for (k=1 , k < m , k++)
                u_k,m-1 ← u_k,m-1 + u_k-1,m-1 ;
step3:      for (k=1 , k < m , k++) in parallel
                for (j=m-2 , j>=0 , j--)
                    u_kj ← u_kj + u_k-1,m-1 ;
}
```

نلاحظ أنه في الخطوة 3 $u_{k-1,m-1}$ قد انتشر على طول السطر k من المعالج في العمود m-1 إلى المعالج في العمود 0 كل معالج يضيفه إلى محتواه ويمرره على جاره الأيسر.

3-4-6 تحليل خوارزمية المجاميع البادئة على الشبكة:

إن كل خطوة تتطلب زمناً O(m) ولذلك فإن $t(n)=O(n^{1/2})$ ، وبما أن عدد المعالجات p(n)=n فإن الكلفة: $c(n)= O(n^{3/2})$ وهي ليست أمثلية.

مثال(3-4):

لتكن n=16 سنوضح عمل الإجراء MESH PREFIX SUMS فيما يلي في الشكل التالي وفق جداول نرمز فيها لشبكة المعالجات وهي تحوي القيم : $A_{ij} = x_i + x_{i+1} +\ldots+ x_j$.

x_0	x_1	x_2	x_3
x_4	x_5	x_6	x_7
x_8	x_9	x_{10}	x_{11}
x_{12}	x_{13}	x_{14}	x_{15}

مبدئياً تحتوي المعالجات على القيم المبينة في الجدول (أ)

X_0	A_{01}	A_{02}	A_{03}
X_4	A_{45}	A_{46}	A_{47}
X_8	A_{89}	$A_{8\,10}$	$A_{8\,11}$
x_{12}	$A_{12\,13}$	$A_{12\,14}$	$A_{12\,15}$

تغير القيم بعد الخطوة 1 تبين في الجدول (ب)

x_0	A_{01}	A_{02}	A_{03}
x_4	A_{45}	A_{46}	A_{07}
x_8	A_{89}	$A_{8\,10}$	$A_{0\,11}$
x_{12}	$A_{12\,13}$	$A_{12\,14}$	$A_{0\,15}$

تغير القيم بعد الخطوة 2 تبين في الجدول (جـ)

x_0	A_{01}	A_{02}	A_{03}
A_{04}	A_{05}	A_{06}	A_{07}
A_{08}	A_{09}	$A_{0\,10}$	$A_{0\,11}$
$A_{0\,12}$	$A_{0\,13}$	$A_{0\,14}$	$A_{0\,15}$

تغير القيم بعد الخطوة 3 تبين في الجدول (د)

الشكل (3-7) يوضح حساب المجاميع البادئة باستعمال الإجراء MESH PREFIX SUMS

لنفترض الآن أن لدينا شبكة من المعالجات $N^{1/2} \times N^{1/2}$ حيث $N<n$. لحساب المجاميع البادئة للسلسلة $X=\{x_0,x_1,..,x_{n-1}\}$ يتلقى كل معالج n/N عنصر من X مبدئياً ويحسب مجاميعها البادئة. يمكن أن نعدل الإجراء السابق MESH PREFIX SUMS مثل خوارزمية الشجرة ولذلك عندما

ينتهي الاجراء نجد أن كل معالج يحتوي على n/N من المجاميع البادئة من X . يأخذ الإجراء المعدل زمناً $O(n/N) + O(n^{1/2})$ وكلفة $O(n/N) + O(n^{3/2})$ ، ونلاحظ أن هذه الكلفة ليست أمثلية عندما يكون: $N = O(n^{3/2})$.

3-4-7 مسألة الأعمال المتسلسلة وفق فترات انتهائها :

ليكن لدينا مجموعة من الأعمال التي نريد تنفيذها على آلة واحدة وليكن عددها n ولنرمز لها كما يلي: $J = \{j_0, j_1, .., j_{n-1}\}$. يمكن للآلة أن تنفذ عملاً واحداً في زمن ما وعندما تحدد عملاً فيجب عليها أن تكمله قبل أن تبدأ بمعالجة العمل التالي.

يرتبط بكل عمل j_i زما يلي:

1- زمن معالجة t_i .
2- مدة انتهاء d_i يجب أن يكتمل العمل من خلالها.

نعرف جدول الأعمال بأنه تبديل الأعمال في J الذي يحدد ترتيب تنفيذها، ويقال عن الجدول أنه مناسب أو عملي إذا كان كل عمل ينتهي بمدة انتهائه .

والمطلوب أنه إذا كان لدينا n عمل $\{j_0, j_1, .., j_{n-1}\}$ لها أزمنة المعالجة $\{t_0, t_1, .., t_{n-1}\}$ ولها فترات الانتهاء $\{d_0, d_1, .., d_{n-1}\}$ فهل يوجد جدول عملي لهذه الأعمال .

يمكن الإجابة عن هذا السؤال بالإيجاب إذا وفقط إذا كان أي جدول أعمال تنفذ فيه الأعمال بترتيب متزايد لفترات الانتهاء عملياً. لذلك يكفي لحل المسألة إيجاد جدول أعمال ترتب فيه الأعمال بترتيب متزايد لفترات الانتهاء وفحص فيما إذا كان هذا الجدول عملياً أم لا وفي حال تم ذلك فإن الإجابة عن السؤال تكون نعم و إلا فالإجابة لا.

تحتاج هذه الخوارزمية لكلفة زمنية $O(n \log n)$ لترتيب الأعمال بشكل تسلسلي ومن ثم إلى كلفة زمنية $O(n)$ لاختبار فيما إذا كان كل عمل يمكن أن يتم في فترة انتهائه.

سنوجد الآن خوارزمية متوازية تعمل على حاسب باتصال شجري ولتكن المعالجات الأوراق هي $p_0, p_1, .., p_{n-1}$ وسنفترض من أجل السهولة أن الأعمال مرتبة بترتيب متزايد لفترات الانتهاء أي أن: $d_0 \le d_1 \le .. \le d_{n-1}$. وليكن المعالج p_i يحتوي مبدئياً على t_i و d_i ، تعطى الخوارزمية كإجراء TREE SEQUENCING.

1- خوارزمية الإجراء TREE SEQUENCING(J,answer):

الخطوة 1: احسب المجاميع البادئة $s_0,s_1,..,s_{n-1}$ لـ $t_0,t_1,...,t_{n-1}$

الخطوة 2:

1- المعالج الورقة p_i: إذا كان $s_i \le d_i$ عندئذ أرسل "نعم" للأب وإلا فأرسل "لا" للأب.

2- المعالج الوسطي: إذا كان الدخل من كلا الولدين هو "نعم" عندئذ أرسل نعم للأب و إلا فأرسل "لا" للأب.

3- المعالج الجذر: إذا كان الدخل من كلا الولدين هو "نعم" عندئذ ضع في answer "يوجد جدول أعمال عملي" و إلا فضع في answer "لا يوجد جدول أعمال عملي".

```
Void TREE SEQUENCING(J,answer)
{
step1:      compute s0,s1,..,sn-1 , the prefix sums of t0,t1,..,tn-1
step2:      (i) leaf processor pi
               if (si <= di )
                  send " yes " to parent ;
              else
                  send " no " to parent ;
            (ii) intermediate processor
               if (inputs from both children are " yes ")
                  send " yes " to parent ;
               else
                  send " no " to parent ;
            (iii)   root processor
               if (inputs from both children are " yes ")
                  answer ← " feasible schedule exists" ;
               else
                  answer ← " no feasible schedule ";
}
```

مثال(3-5):

لتكن n=4 حيث $\{t_0,t_1,..,t_{n-1}\} = \{1,2,3,4\}$ و $\{d_0,d_1,..,d_{n-1}\} = \{3,5,7,9\}$. لذلك فإن:

$\{s_0,s_1,..,s_{n-1}\} = \{1,4,7,11\}$.

لدينا : $s_0 \le d_0$ و $s_1 \le d_1$ و $s_2 \le d_2$ و $s_3 > d_3$ وبالتالي لا يوجد جدول أعمال عملي لهذه المسألة.

2- تحليل خوارزمية الإجراء TREE SEQUENCING:

تتطلب كلاً من الخطوتين 1 و 2 : $O(\log n)$ عملية. ومنه نجد أنه يتم التحكم بزمن تنفيذ الخوارزمية من خلال الزمن المطلوب للترتيب المبدئي للأعمال في المعالجات الأوراق وفق الترتيب المتزايد لفترات الانتهاء وهو $\Omega(n)$.

8-4-3 مسألة حقيبة الظهر THE KNAPSACK PROBLEM:

[MLFi,1981], [WShi,1979] , [DFGP,1982]

في هذه المسألة تتميز حقيبة الظهر بأنها تستطيع حمل وزن أعظمي W ومجموعة من n مادة ولتكن: $A=\{a_0,a_1,..,a_{n-1}\}$ والتي أوزانها على التوالي: $\{w_0,w_1,..,w_{n-1}\}$ كما يرتبط بكل مادة فائدة ونرمز لمجموعة الفوائد بـ: $\{p_0,p_1,..,p_{n-1}\}$، إذا وضعنا في حقيبة الظهر الجزء z_i من المادة التي وزنها w_i حيث $0 \le z_i \le 1$ عندئذ فإن الفائدة المكتسبة هي $z_i p_i$. والمطلوب في هذه المسألة هو تعبئة حقيبة الظهر بالمواد أو بأجزاء منها بحيث:

1- الوزن الناتج للمواد المحددة لا يتجاوز W
2- الفائدة المكتسبة الناتجة أكبر ما يمكن.

لدينا 2n+1 عدد موجب معطى: $w_0,w_1,..,w_{n-1},W,\ p_0,p_1,..,p_{n-1}$ وهي ضرورية من أجل زيادة الكمية:

$$Q = \sum_{i=0}^{n-1} z_i \times p_i$$

وهي تخضع للشرطين التاليين:

1. أياً كان i فإن: $0 \le z_i \le 1$
2. $\sum_{i=0}^{n-1} z_i \times w_i \le W$

إن الحل الأمثل هو سلسلة $Z = \{z_0,z_1,..,z_{n-1}\}$ يرفع Q للحد الأعلى ويستوفي الشروط 1و2 ويمكن أن نحصل على مثل هذا الحل إذا كانت المواد المختبرة مرتبة بشكل متزايد للنسب p_i/w_i . إذا كانت

المادة التي نختبرها توافق الجزء المتبقي من حقيبة الظهر عندئذ نأخذ المادة و إلا نأخذ جزء منها فقط ونضعه في حقيبة الظهر. يتطلب هذا العمل تسلسلياً كلفة زمنية قدرها $O(n \log n)$ لترتيب الفوائد والأوزان ومن ثمَّ كلفة زمنية $O(n)$ لفحص جميع المواد واحدة في كل مرة.

سنأخذ الآن خوارزمية متوازية لإيجاد السلسلة الأمثلية $\{z_0,z_1,..,z_{n-1}\}$ تعمل على حاسب متوازي متصل شجرياً يحتوي على المعالجات الأوراق $p_0,p_1,..,p_{n-1}$.

وسنفترض من أجل سهولة ترميز العبارات الأصلية في المسألة أن المواد مرتبة بشكل متناقص لنسب الفوائد إلى الأوزان أي أن: $p_0/w_0 \geq p_1/w_1 \geq \ldots \geq p_{n-1}/w_{n-1}$ ، يحتوي المعالج الورقة P_i في البداية على w_i و p_i و W. تعطى الخوارزمية فيما يلي كإجراء TREE KNAPSACK

1- خوارزمية الإجراء TREE KNAPSACK:

الخطوة 1:

1- احسب احسب المجاميع البادئة $s_0,s_1,..,s_{n-1}$ لـ $w_0,w_1,..,w_{n-1}$

2- من أجل $i=1$ إلى $n-1$ نفذ على التوازي: كل معالج P_i يحسب s_{i-1} .

الخطوة 2:

من أجل $i=0$ إلى $n-1$ نفذ على التوازي :

إذا كان $s_i \leq W$ عندئذ: ضع $z_i \leftarrow 1$

و إلا: إذا كان $(s_i > W)$ و $(s_{i-1} \leq W)$ عندئذ: $z_i \leftarrow (W-s_{i-1})/w_i$

و إلا: $z_i \leftarrow 0$.

```
Void TREE KNAPSACK(A,W,Z)
{
Step1:      (1.1) compute s0,s1,..,sn-1 , the prefix sums of w0,w1,..,wn-1
            (1.2) for (i=1 , i < n , i++) in parallel
                        Pi computes si-1 ;
Step2:      for (i=0 , i < n , i++) in parallel
                  if (si ≤ W )
                        zi ←1 ;
                  else if ((si > W) && (si-1 ≤ W))
                        zi←(W-si-1)/wi ;
                  else
                        zi ← 0 ;
}
```

نلاحظ أنه يمكن حساب الجزء الناتج في الجذر كما يلي:

1- كل معالج ورقة P_i يحسب الجزء (i) وهو $z_i \times p_i$ ويرسل الجزء (i) إلى والده.
2- كل معالج وسطي يجمع إدخاليه الواصلين من ولديه ويرسل النتيجة إلى والده.
3- المعالج الجذر يجمع إدخاليه اللذين تلقاهما من ولديه وهذا هو Q.

مثال(3-6):

لتكن $n = 4$ و $\{w_0,w_1,..,w_{n-1}\} = \{5,9,2,4\}$ و $\{p_0,p_1,..,p_{n-1}\} = \{100,135,26,20\}$ و $W=15$. ومنه نجد أن: $\{s_0,s_1,..,s_{n-1}\} = \{15,14,16,20\}$ وبما أن : $s_0 \le W$ و $s_1 \le W$ فإن: $z_0= z_1=1$ وأيضاً: $s_2 > W$ لذلك: $z_2 = (15-14)/2 = 0.5$ وأخيراً : $s_3 > W$ لذلك: $z_3 = 0$.

2- تحليل خوارزمية حقيبة الظهر:

تحتاج الخطوتان 1 و 2 : O(log n) و O(1) خطوة على التوالي. كما يتحكم بزمن تنفيذ هذه الخوارزمية الزمن المأخوذ مبدئياً من أجل ترتيب الأجزاء والأوزان والأوراق بشكل تناقص نسبها وهذا الزمن هو $\Omega(n)$.

3- حلول مسألة حقيبة الظهر على الشبكة:

رأينا أن الحلول الشجرية تحتاج على الأقل زمناً $\Omega(n)$ وذلك إذا لم تكن السلسلة المدخلة مرتبة أصلاً. وسنبين الآن كيف أنه في هذه الحالات يكون الحاسب المتوازي باتصال شبكي نموذجاً أكثر فعالية لحل مسائل الأمثلية والقرار هذه.

لنفترض أن الدخل في مسائل حقيبة الظهر والأعمال المتسلسلة ليس مرتباً كما هو مطلوب في الإجراء TREE KNAPSACK و TREESEQUENCING على التوالي.

إذا كان لدينا حاسب شبكي $n^{1/2} \times n^{1/2}$ عندئذ:

1- يمكن أن ترتب سلسلة الدخل بـ n عنصر على الشبكة بزمن $O(n^{1/2})$

2- يمكن أن يعمل كل من الإجراءين السابقين بسهولة على الشبكة بزمن $O(n^{1/2})$

وهذا يعني أن الزمن الكلي المطلوب للتنفيذ لكل من مسألتي الأعمال المتسلسلة وحقيبة الظهر هو: $t(n) = O(n^{1/2})$ وهذا أسرع من الزمن المطلوب باستعمال الخوارزميات الشجرية.

بما أن $p(n) = n$ عندئذ نجد: $c(n) = O(n^{3/2})$ وهذه الكلفة ليست أمثلية من وجهة نظر زمن التنفيذ $O(n \log n)$ الكافي لحل المسألتين تسلسلياً.

لنفترض الآن أن لدينا شبكة $N^{1/2} \times N^{1/2}$ حيث: $N < \log^2 n$ ونعلم أن شبكة بهذا العدد من المعالجات يمكن أن ترتب سلسلة بـــ n عنصر بكلفة أمثلية $O(n \log n)$. وبما أن $\log^2 n < n^{2/3}$ من أجل n كبير كفاية فإن المجاميع البادئة لسلسلة بـــ n عنصر يمكن أن تحسب على شبكة $N^{1/2} \times N^{1/2}$ بكلفة أمثلية $O(n)$. هاتان العمليتان المسميتان الترتيب وحساب المجاميع البادئة تتحكمان في حل مسألتي الأعمال المتسلسلة و حقيبة الظهر، وهذا يعني أنه يمكن حل المسألتين بشكل أمثلي على شبكة من المعالجات.

5-3 خوارزميات متوازية لأجهزة متعددة المعالجات ذات ذاكرة مشتركة:

سنقوم فيما يلي بعرض بعض المسائل الشهيرة رياضياً ، ونضمن التعاريف الخاصة بها، ونضع خوارزميات الحلول التسلسلية والمتوازية لها ، ونميز بين الخوارزميات المتوازية من أجل النماذج المختلفة للحواسيب وطرائق اتصالها في بعض منها ، كما نضع الشروط المناسبة لحلها والخواص الهامة التي يجب أن تحققها ، ومن ثم نعرض تحليلاً لكل منها لدراسة أمثليتها وكلفتها الزمنية وغيرها مع ذكر بعض الأمثلة التوضيحية التي تساعد على فهم المسألة. كما نذكر الخواص والأهداف التي نحاول الوصول إليها في كل خوارزمية.

في البداية سنأخذ مسألتين من مسائل المقارنة مسألة الاختيار في سلسلة التي ظهرت في العديد من تطبيقات علوم الحاسبات والإحصاء ونحاول إيجاد خوارزمية متوازية لحل هذه المسألة تحقق عدداً من الأهداف والخواص الضرورية، ومن ثم نأخذ مسألة أخرى تعرف بمسألة الدمج (الدمج متسلسلتين من البيانات لتشكيل متسلسلة ثالثة جديدة) وهي أحد مسائل المقارنة فقد ازداد الدمج في علم الحاسبات مع تنوع المحتويات متضمناً قواعد البيانات بشكل خاص وترتيب الملفات عموماً إذ أن العديد من هذه التطبيقات تستعمل مسألة الدمج للبيانات الرقمية ومن الضروري توظيف الدمج بشكل تام بحيث يحذف التكرار في الإدخالات من المتسلسلة الناتجة.

1-5-3 مسألة الاختيار:

تحتوي دراستنا عن تحليل وتصميم الخوارزميات على المسألة التالية:

لنأخذ سلسلة S من n عنصر وعدد صحيح k حيث $1<k<n$ والمطلوب تحديد العنصر k الأصغر في S //تعرف هذه المسألة بمسألة الاختيار//.

لقد ظهرت هذه المسألة في العديد من التطبيقات في علوم الحاسبات والإحصاء. وغرضنا هنا هو إيجاد الخوارزمية المتوازية لحل هذه المسألة على حواسيب من النموذج SIMD بذاكرة مشتركة. وسنصمم الخوارزمية بحيث تلائم عدداً من الأهداف ومن ثم نحللها لنتحقق من أن تلك الأهداف ملائمة فعلاً.

تنتمي هذه المسألة إلى مجموعة مسائل تدعى مسائل المقارنة /comparison problems/ حيث تحل هذه المسائل عادة عن طريق مقارنة أزواج من العناصر في سلسلة مدخلة.

بداية لنأخذ التعاريف التالية:

1-الترتيب الخطي Linear Order:

نقول عن عناصر مجموعة A أنها تشكل ترتيباً خطياً إذا وفقط إذا كان:

1. من أجل أي عنصرين a و b من A فإن: a يسبق b ، أو a ينطبق على b ، أو b يسبق a.
2. من أجل أية ثلاثة عناصر a و b و c فإنه إذا كان a يسبق b و b يسبق c فإن a يسبق c.

وكمثال نجد أن مجموعة جميع الأعداد الصحيحة تشكل ترتيباً خطياً، مثال آخر مجموعة الحروف في الأبجدية اللاتينية. فهذه المجموعات هي تراتيب خطية.

2-الرتبة (الدرجة) Rank:

إذا كان لدينا متتالية $S=\{s_1,s_2,..,s_n\}$ عناصرها تنتمي إلى مجموعة تشكل ترتيباً خطياً. تعرف رتبة العنصر s_i من S بأنها تساوي: ((عدد العناصر في S التي تسبق s_i) + 1).

ومنه: من أجل $S=\{8,-3,2,-5,6,0\}$ رتبة 0 هي 3.

ملاحظة: إذا كان $s_i = s_j$ فإن s_i يسبق s_j إذا وفقط إذا كان $i < j$.

3-الاختيار Selection:

إذا كان لدينا متتالية $S=\{s_1,s_2,..,s_n\}$ عناصرها تنتمي إلى مجموعة تشكل ترتيباً خطياً ولنأخذ k عدد صحيح حيث $1 \le k \le n$. والمطلوب تحديد العنصر الذي رتبته تساوي k.

مرة أخرى في المجموعة : $S=\{8,-3,2,-5,6,0\}$ العنصر من الرتبة 4 هو 2.

سنرمز للعنصر من الرتبة k بـــ $s_{(k)}$ ، وسنفترض في المناقشة التالية أن S متتالية من الأعداد الصحيحة كما في المثال السابق ، وعندها فإن الاختيار يكون من أجل إيجاد العنصر الأصغر k.

لنأخذ مجموعة الرموز التالية:

من أجل عدد حقيقي r فإن:

$\lceil r \rceil$ ترمز إلى أصغر عدد صحيح أكبر من r أو يساوي r (the ceiling of r)

$\lfloor r \rfloor$ ترمز إلى أكبر عدد صحيح أصغر من r أو يساوي r (the floor of r)

لذلك فإن: $3=\lceil 3.0 \rceil=\lfloor 3.0 \rfloor$, $3=\lfloor 3.9 \rfloor$, $4=\lceil 3.1 \rceil$

4-التعقيد Complexity:

هناك ثلاث قيم خاصة في مسألة الاختيار يمكن تمييزها بسهولة وهي: $k=n/2$, $k=n$, $k=1$. نستطيع في الحالتين الأولى والثانية أن نبحث عن العناصر الأصغر والأكبر في S على التوالي. أما في الحالة الثالثة ستكون $s_{(k)}$ في وسط S وبالتالي فإن نصف العناصر أصغر منه (أو مساوية له) والنصف الآخر أكبر منه (أو مساوية له). ومن الواضح على الأقل في نموذج الحسابات التسلسلي أن الحالتين الأولى والثانية أسهل في الحل من الحالة $k=n/2$ أو أي قيمة أخرى. فمن أجل $k=1, k=n$ نقوم بفحص العنصر التالي باستخدام عنصر ما نحفظ فيه العنصر الأصغر (أو الأكبر) الذي نجده حتى نحصل على النتيجة.

أما من أجل الحالة $(1 < k < n)$ فإنه لا يوجد حلول واضحة كالحالتين الباقيتين. ومن الواضح أنه إذ كانت S مرتبة حيث: $S=\{s_{(1)},s_{(2)},..,s_{(n)}\}$ فإن مسألة الاختيار تصبح سهلةً وعاديةً ، وعندها نستطيع الحصول على $s_{(k)}$ في خطوة واحدة. طبعاً لا نفترض أن الحالة كذلك ولكننا نرغب بترتيب S

ومن ثم نلتقط العنصر الأصغر k وبهذا الشكل نستطيع حل مسألة الاختيار من أجل جميع قيم k وليس فقط من أجل حالة واحدة خاصة وذلك من أجل قيم كبيرة لـــ n.

وبغض النظر عن قيمة k نجد أنه عند تحديد العنصر الأصغر k يجب علينا أن نفحص كل عنصر في S على الأقل مرة واحدة. وهذا يقودنا إلى الحد الأدنى لعدد الخطوات التسلسلية المطلوبة لحل المسألة وهو $\Omega(n)$ أي أن الحد الأدنى للكلفة لأي خوارزمية متوازية للاختيار هو $\Omega(n)$.

3-5-1-1 الخوارزمية التسلسلية لمسألة الاختيار :

هناك سببين لاهتمامنا بالخوارزمية التسلسلية الأول هو أن الخوارزمية المتوازية تؤسس على الخوارزمية التسلسلية كما أنها تعتبر التنفيذ المتوازي للخوارزمية التسلسلية على حواسيب EREW SM SIMD والثاني أن الخوارزمية المتوازية تفترض وجود الخوارزمية التسلسلية وتستعملها كإجراء.

فيما يلي نعرض الخوارزمية وهي على شكل إجراء تسلسلي SEQUENTIAL SELECT يتعاود طبيعياً وهي تستعمل طريقة فرق تسد (divide and conquer approach) لتصميم الخوارزمية.

الإدخالات البدائية للإجراء هي المتسلسلة S والعدد الصحيح k . في كل مرحلة من التعاودية يطرح عدد من عناصر S جانباً من الاهتمام لبدء تحديد العنصر الأصغر k وهذا يستمر حتى يحدد العنصر k في النهاية.

نرمز بـــ |S| لندل على حجم المتسلسلة S وهكذا نجد مبدئياً: |S| = n. وليكن Q ثابت صحيح صغير يحدد لاحقاً عند تحليل زمن تنفيذ الخوارزمية.

1- خوارزمية الإجراء SEQUENTIAL SELECT (S, *k*)

الخطوة 1: – إذا كان |S|<Q عندها رتب S وأعد العنصر k مباشرة

– وإلا قسم S إلى |S|/Q متسلسلة جزئية كل منها يحوي Q عنصر على الأكثر.

الخطوة 2: – رتب كل من المتسلسلات الجزئية وحدد العنصر الأوسط في كل منها.

الخطوة 3: – استدعي الإجراء SEQUENTIAL SELECT تعاودياً لإيجاد m العنصر الأوسط من المتوسطات في المجموعات |S|/Q (التي أوجدت في الخطوة2).

الخطوة 4: - أنشئ ثلاث متسلسلات جزئية S_1 و S_2 و S_3 من العناصر الأصغر من m والمساوية لـــ m والأكبر من m على التوالي.

الخطوة 5: - إذا كان $|S_1| \geq k$ عندها [يجب أن يوجد العنصر k في S_1] استدعي الإجراء SEQUENTIAL SELECT تعاودياً لإيجاد العنصر الأصغر k في S_1.

- وإلا إذا كان $|S_1| \geq k$ عندها أعد m .

- وإلا استدعي الإجراء SEQUENTIAL SELECT تعاودياً لإيجاد العنصر الأصغر $(k - |S_1| - |S_2|)$ في S_3.

```
Void SEQUENTIAL SELECT (S, k)

{

step1:    if    (|S|<Q ) sort S and return the kth element directly ;

          else  subdivide S into |S|/Q subsequences of Q elements each (with
                up to Q-1 leftover elements) ;

step2:    sort each subsequence and determine its median ;

step3:    call SEQUENTIAL SELECT recursively to find m , the median of
          the |S|/Q medians found in step2 ;

step4:    creat three subsequences S1 , S2 , and S3 of elements of S smaller
          than, equal to, and larger than m, respectively ;

step5:    if  (|S1| ≥ k)          /* the kth element of S must be in S1 */

                call SEQUENTIAL SELECT recursively to find the kth
                element of S1

          else if (|S1|+|S2| ≥ k)

                return m ;

              else    call SEQUENTIAL SELECT recursively to find the (k
                      - |S1| - |S2|)th element of S3 ;

}
```

نلاحظ أن العبارة السابقة للإجراء SEQUENTIAL SELECT التي لا تحدد العنصر الأصغر k في S تتعاود في الواقع. ويمكن تطبيق ذلك بطريقة وحيدة وهي أن نأخذ وسيط إضافي في رأس الإجراء وليكن x بالإضافة إلى S و k فنعيد قيمة العنصر الأصغر k في x. وهناك طريقة أخرى أبسط وهي إعادة العنصر الأصغر k كأول عنصر في المتسلسلة S.

2- *تحليل* خوارزمية الإجراء SEQUENTIAL SELECT:

سنعرض الآن تحليل زمن تنفيذ الخوارزمية SEQUENTIAL SELECT: t(n) خطوة بخطوة.

الخطوة 1: طالما أن Q ثابت رتب S عندما |S|<Q وهذا يأخذ زمناً ثابتاً و إلا يتطلب تقسيم S إلى زمن قدره c_1n من أجل ثابت ما c_1.

الخطوة 2: طالما أن كل من المتسلسلات |S|/Q من العناصر Q ثابتة فإنه يمكن ترتيبها بزمن ثابت لذلك فإن الزمن الذي نحتاجه من أجل هذه الخطوة هو c_2n من أجل ثابت ما c_2.

الخطوة 3: لدينا متوسطات عددها |S|/Q إذاً الزمن الذي تأخذه التعاودية هو t(n/Q).

الخطوة 4: يوجد طريقة واحدة لإيجاد m خلال إنشاء S لـ S_1,S_2,S_3 لذلك فإن هذه الخطوة تتم خلال زمن قدره c_3n من أجل ثابت ما c_3.

الخطوة 5: طالما أن m هو المتوسط في عناصر |S|/Q فهناك عدد من العناصر قدره |S|/2Q أكبر منه أو مساوية له . كل من عناصر |S|/Q هي نفسها المتوسط لمجموعة من العناصر Q وهذا يعني أنها تملك عدد من العناصر قدره Q/2 أكبر منها أو مساوية لها.

وهذا يعني أن عدد من العناصر قدره: $(Q/2) \times (|S|/2Q) = |S|/4$ في S من المضمون أن تكون أكبر من m أو مساوية لـ m. وبالتالي : $|S_1| \leq 3|S|/4$ وباستنتاج مشابه فإن: $|S_3| \leq 3|S|/4$

في هذه الخطوة يتم الاستدعاء التعاودي للإجراء SEQUENTIAL SELECT للبحث إما في S_1 أو في S_3 عن العنصر الأصغر وهذا يعني أن الاستدعاء التعاودي يتطلب زمن تابع لحجم المجموعة S_3 و S_1 وهو الذي قمنا بحسابه($\leq 3|S|/4$) ، أي 3n/4 ومنه نجد:

يتطلب الاستدعاء التعاودي في هذه الخطوة للإجراء SEQUENTIAL SELECT زمن قدره t(3n/4).

من التحليل السابق لدينا : $c_4 = c_1 + c_2 + c_3$; $t(n) = c_4n + t(n/Q) + t(3n/4)$

الآن لنعين Q : إذا اخترنا Q كما يلي: $n/Q + 3n/4 < n$ عندها ينفذ الاستدعاءين التعاوديين في الإجراء بتتابع شديد الانخفاض. وإن أي قيمة لـــ $Q \geq 5$ ستسبب ذلك .

لنأخذ $Q=5$ ومنه نجد: $t(n)=c_4n+t(n/5)+t(3n/4)$

وهذه التعاودية يمكن حلها بالافتراض التالي : $t(n) \leq c_5n$ من أجل ثابت ما c_5.

وبالتالي: $t(n) \leq c_4n + c_5(n/5) + c_5(3n/4) = c_4n + c_5\ (19n/20)$

أخيراً لنأخذ : $c_5 = 20\ c_4$ ومنه نجد: $t(n) \leq c_5(n/20) + c_5(19n/20) = c_5n$

وهذا يؤكد افتراضنا أن : $t(n) = O(n)$ الذي افترضناه في عرض الحد الأدنى.

3- الخصائص المطلوبة في الخوارزمية المتوازية للاختيار :

هناك خصائص هامة نرغب في الحصول عليها في الخوارزمية المتوازية وهي التالية:

1. عدد المعالجات:

الخاصية الأولى هي عدد المعالجات المستخدمة في الخوارزمية. وبفرض n هو حجم المسألة التي نريد حلها فإن:

أ- يجب أن تكون P(n) أصغر من n : فمن غير الواقعي أن نفترض أن المعالجات أكثر من مفردات المعلومات خاصة من أجل قيم كبيرة جداً لـــ n. ولذلك من المهم أن تكون P(n) تعبر عن دالة خطية في n ، وهذا يعني:

$$P(n) = n^x \ ;\ 0 < x < 1$$

ب- يجب أن تتكيف الخوارزمية مع P(n): في الحساب بشكل عام وفي الحساب المتوازي بشكل خاص نحتاج إلى قوة حسابية إضافية فقد تُطرح مسائل أكبر وأكثر تعقيداً أكثر مما كان محتملاً قبل ذلك. وهذه المسائل الأكبر قد تجعل الخوارزمية عديمة النفع تماماً. وتستخدم الخوارزميات عدد المعالجات على أنها دالة خطية في n مثل $\log n\ ,\ n^{1/2}$ و قد تكون غير مرضية بسبب عدم مرونتها.

ما نحتاجه هو خوارزمية تمتلك الذكاء لكي تتكيف مع عدد المعالجات الحقيقي المتوفر على الحاسب الموجود.

2. زمن التنفيذ:

الخاصية الثانية من الخاصيتين المهمتين هو زمن تنفيذ أسوأ حالة للخوارزمية المتوازية.

أ- يجب أن تكون t(n) صغيرة : الدافع الأساسي لبناء الخوارزمية المتوازية هو تسريع العمليات الحسابية. ولذلك من المهم أن تكون الخوارزميات المتوازية التي نصممها أسرع بشكل مميز من أفضل خوارزمية تسلسلية للمسألة التي بين أيدينا.

ب- يجب أن تتكيف الخوارزمية مع t(n) : فمثلاً قد يرغب شخص ما أن يملك خوارزمية يتناقص زمن تنفيذها كلما كان عدد المعالجات المستعملة أكبر. فمن المرغوب أن t(n) تتغير بشكل متناسب (عكسياً) مع p(n) بدون مجموعة الحدود لـ p(n).

4. الكلفة :

أخيراً نرغب بالحصول على خوارزميات متوازية يتحقق فيها :

$$c(n) = p(n) \times t(n)$$

والتي تنسب الحد الأدنى المعروف للخوارزمية على عدد العمليات التسلسلية المطلوبة في أسوأ حالة لحل المسألة. وبمعنى آخر الخوارزمية المتوازية ستكون بأفضل كلفة.

إن اجتماع الخواص السابقة عادة يكون صعباً وفي بعض الأحيان مستحيلاً. وبشكل خاص عندما تكون المعالجات المتعددة موصولة عن طريق شبكة اتصالات حيث أن الشكل الهندسي للشبكة يفرض حدوداً على ما نستطيع إتمامه عن طريق الخوارزمية المتوازية. ولكن الأمر مختلف تماماً عندما تكون الخوارزمية منفذة على حاسب متوازي بذاكرة مشتركة ، فلا نهتم كثيراً هنا بهذه الخواص إذ أن النموذج يكون قوياً ومرناً.

بالنسبة للخوارزمية المتوازية التي سنصفها بعد قليل من أجل اختيار العنصر الأصغر k في المتسلسلة S={s1,s2,..,sn} ، والتي ستنفذ الخوارزمية على حاسب EREW SM SIMD مع N معالج حيث N<n. نجد أن الخوارزمية تتمتع بجميع الخواص السابقة:

1- إنها تستعمل المعالجات $p(n) = n^{1-x}$ حيث: $0<x<1$ ويمكن الحصول على قيمة x من العلاقة: $N= n^{1-x}$. لذلك فإن p(n) خطية ومتكيفة sublinear and adaptive.

2- إنها تنفذ في زمن $t(n) = O(n^x)$ حيث x تعتمد على عدد المعالجات المتوفرة على حاسب متوازي ويمكن الحصول عليها من العلاقة السابقة. لذلك فإن t(n) أصغر من زمن تنفيذ أفضل خوارزمية تسلسلية التي ذكرناها. كما أن الخوارزمية متكيفة مع زمن التنفيذ t(n) فإذا كان الأكبر p(n) فإن الأصغر t(n) والعكس بالعكس.

3- إنها تملك كلفة $C(n) = n^{1-x} \times O(n^x) = O(n)$ وهي الأفضل نظراً للحد الأدنى الذي ذكرناه سابقاً.

أخيراً عند التعامل مع القيم الحقيقية للأنواع المذكورة (مثلاً: n^{1-x} , n^x) ستكون مدورة إلى أعداد صحيحة مناسبة لذلك فإن التمثيل الحقيقي لـــ n^{1-x} عدد المعالجات المستخدمة في الخوارزمية سيفسر كـــ $\lfloor n^{1-x} \rfloor$ وهذا من أجل أن يضمن أن النتيجة الصحيحة ليست متجاوزة العدد الحقيقي للمعالجات. وبالعكس التمثيل الحقيقي لـــ n^x زمن تنفيذ أسوأ حالة للخوارزمية سيفسر بـــ $\lceil n^x \rceil$ وهذا ليضمن أن النتيجة الصحيحة ليست أصغر من الزمن الحقيقي لتنفيذ أسوأ حالة.

4- خوارزمية الاختيار المتوازي

سندرس الآن خوارزمية الاختيار المتوازي على حاسب EREW SM SIMD وهي معطاة بالإجراء PARALLEL SELECT، ولنفترض ما يلي:

1- لتكن $S=\{s_1,s_2,..s_n\}$ متسلسلة من الأعداد الصحيحة ، و k عدد صحيح حيث: $1\leq k\leq N$ ، والمطلوب تحديد العنصر الأصغر k في المتسلسلة S وهذه هي الإدخالات البدائية للإجراء PARALLEL SELECT.

2- الحاسب المتوازي يتكون من N معالج $p_1,p_2,..,p_N$.

3- كل معالج تلقى n وحسب x من العلاقة $N = n^{1-x}$ حيث $0 < x < 1$.

4- كل من المعالجات n^{1-x} لها القدرة على ترتيب متسلسلة من n^x عنصر في ذاكرته المحلية .

5- كل معالج يستطيع تنفيذ الإجراء: ALLSUMS و BROADCAST و SEQUENTIAL SELECT

6- M هي مصفوفة في الذاكرة المقسمة طولها N ورمزها هو M(i).

خوارزمية الإجراء PARALLEL SELECT(S,k) :

الخطوة1 : إذا كان $|S| \leq 4$ عندها يستعمل p_1 خمس مقارنات على الأغلب لإعادة العنصر k

وإلا:

1. قسّم المتسلسلة S إلى $|S|^{1-x}$ متسلسلة جزئية ، طول المتسلسلة S_i هو $|S|^x$ حيث: $1 \leq i \leq |S|^{1-x}$ ؛
2. المتسلسلة الجزئية S_i محددة للمعالج p_i .

الخطوة 2: من أجل i=1 إلى $|S|^{1-x}$ نفذ على التوازي:

1. [يحصل كل معالج p_i على العنصر الأوسط m_i ؛ العنصر $\lceil |S_i|/2 \rceil$ ؛ في مجموعته الجزئية المحددة] استدعي إجراء الاختيار التسلسلي من أجل القيم التالية: SEQUENTIAL SELECT(S_i , $\lceil |S_i|/2 \rceil$)
2. يخزن المعالج p_i العنصر الأوسط m_i في المصفوفة M_i.

الخطوة 3: [يستدعي الإجراء نفسه تعاودياً للحصول على العنصر الأوسط m من M]
PARALLEL SELECT (M, $\lceil |M|/2 \rceil$)

الخطوة 4: تُقَسَم المتسلسلة S إلى ثلاث متسلسلات جزئية:

L = { si € S : si < m }
E = { si € S : si = m }
G = { si € S : si > m }

الخطوة 5: إذا كان $|L| \geq k$ عندها استدعي الإجراء PARALLEL SELECT (L,K)

وإلا إذا كان $|L|+|E| \geq k$ عندها أعد العنصر m

وإلا استدعي الإجراء PARALLEL SELECT (G,K-|L|-|E|).

```
Void PARALLEL SELECT(S,k)

{

Step1:  if (|S| ≤ 4)

        p1 uses at most five comparasions to return the kth element ;

    else
```

(i) S is subdivided into $|S|^{1-x}$ subsequences S_i of length $|S|^x$ each, where $1 \le i \le |S|^{1-x}$, and

(ii) Subsequences S_i is assigned to processor p_i ;

```
Step2:  for (i=1 , i<=|S|^(1-x) ,i++) in parallel
        {
        (2.1) /* p_i obtains the median m_i , i.e., the ⌈|S_i|/2⌉th element, of its
              associated subsequence */

              SEQUENTIAL SELECT(S_i , ⌈|S_i|/2⌉) ;

        (2.2) p_i stores m_i in M_i ;

        }

Step3:   /* the procedure is called recursively to obtain the median m of M
         */

        PARALLEL SELECT (M, ⌈|M|/2⌉) ;

Step4:  the sequence S is subdivided into three subsequences:

        L = { si € S : si < m },
        E = { si € S : si = m }, and
        G = { si € S : si > m }.
Step5:  if (|L| ≥ k)
             PARALLEL SELECT (L,K) ;
        else if (|L|+|E| ≥ k)
             return m ;
        else
             PARALLEL SELECT (G,K-|L|-|E|) ;
}
```

تحليل خوارزمية الإجراء PARALLEL SELECT(S,k) :

كالمعتاد سنرمز للزمن الذي يحتاجه الإجراء من أجل n إدخال بـــ t(n):

الخطوة 1: يحتاج كل معالج لإنجاز هذه الخطوة إلى العنوان البدائي A للمتسلسلة S في الذاكرة المشتركة وإلى حجمها |S| وإلى قيمة k. هذه الكميات يمكن نشرها إلى جميع المعالجات باستعمال

الإجراء BROADCAST وهذا يتطلب الزمن : $O(\log n^{1-x})$ ، فإذا كان $|S| \le 4$ عندها المعالج p_1 يعيد العنصر k في زمن ثابت وإلا p_i يحسب عنوان أول وآخر عنصر في S_i من العلاقتين: $[\ A + i\, n^x - 1\ \ ,\ A + (i-1)\, n^x\ ]$ على التوالي. ويمكن تنفيذ ما سبق في زمن ثابت لذلك فإن الخطوة 1 تأخذ $c_1 \log n$ من وحدات الزمن من أجل ثابت ما c_1 .

الخطوة 2: يقوم الإجراء SEQUENTIAL SELECT بإيجاد العنصر الأوسط لمتسلسلة طولها n^x في $c_2 n^x$ من وحدات الزمن من أجل ثابت ما c_2.

الخطوة 3: طالما أننا نستدعي الإجراء PARALLEL SELECT من أجل متسلسلة بطول n^{1-x} فإن هذا يتطلب زمن هو $t(n^{1-x})$.

الخطوة 4: يمكن أن تقسم المتسلسلة S إلى L,E,G كما يلي:

1- يُنشَر m إلى جميع المعالجات في زمن $O(\log n^{1-x})$ مستخدماً الإجراء BROADCAST.

2- كل من المعالجات p_i قسم S_i إلى ثلاث مجموعات متسلسلات جزئية L_i , E_i , G_i من العناصر الأصغر من m والمساوية لـ m والأكبر من m على التوالي. ويمكن لهذا أن يتم في زمن أسي في حجم S_i وهذا الزمن هو $O(n^x)$.

3- تدمج المتسلسلات الجزئية L_i , E_i , G_i في الشكل L , E , G . وسنعرض كيف يمكن أن يتم هذا من أجل L_i وبإجراءات مماثلة وبنفس زمن التنفيذ يمكن أن نستنتج دمج E_i , G_i على التوالي.

لتكن $a_i = |L_i|$ ، من أجل كل i حيث $1 \le i \le n^{1-x}$ فإن المجموع يحسب من العلاقة:

$$z_i = \sum_{j=1}^{i} a_j$$

يمكن أن نحصل على جميع هذه المجاميع من n^{1-x} معالج في زمن $O(\log n^{1-x})$ وذلك باستخدام الإجراء ALLSUMS.

لتكن $z_0 = 0$ ، تقوم جميع المعالجات بدمج متسلسلاتها الجزئية بآن واحد بالشكل L كما يلي: يقوم المعالج P_i بنسخ L_i في L مبتدئاً بالموقع $z_{i-1} + 1$ ويمكن أن يتم ذلك بزمن قدره $O(n^x)$. ولذلك فإن الزمن المطلوب في هذه الخطوة هو $c_3 n^x$ من أجل ثابت ما c_3.

الخطوة 5: حصلنا على حجم L الذي نحتاجه في هذه الخطوة من الخطوة السابقة من خلال حساب Z_n^{1-x} . وكذلك بالنسبة لحجمي E , G. ويجب علينا الآن تحديد الزمن المستهلك من كلا الخطوتين التعاوديتين. طالما أن m هي العنصر الأوسط في M فإنه لدينا $n^{1-x}/2$ عنصر هي بالتأكيد أكبر منه. علاوة على ذلك فإن كل عنصر من M أصغر على الأقل من $n^x/2$ عنصر في S. لذلك $|L| \leq 3n/4$ وبشكل مشابه فإن $|G| \leq 3n/4$ وبالتالي فإن الخطوة 5 تتطلب على الأغلب زمن $t(3n/4)$.

إن التحليل السابق ينتج التعاودية التالية لـــ $t(n)$:

$$T(n) = c_1 \log(n) + c_2 n^x + t(n^{1-x}) + c_3 n^x + t(3n/4)$$

والتي حلها هو $t(n) = O(n^x)$ وذلك من اجل $n > 4$

طالما أن $p(n) = n^{1-x}$ فإنه لدينا:

$$c(n) = p(n) \times t(n) = n^{1-x} \times O(n^x) = O(n)$$

وهي الكلفة الأفضل المستنتجة في عرض الحد الأدنى سابقاً.

ومن ناحية أخرى نلاحظ أن n^x أكبر من log n من أجل أي x (وقد استعملنا هذه الحقيقة عند تحليلنا للإجراء PARALLEL SELECT) وطالما أن $N = n^{1-x}$ و $n/n^x < n/\log n$ فهذا يعطي أن الإجراء PARALLEL SELECT له أفضل كلفة حيث $N < n/\log n$

مثال (3−7): يوضح عمل الإجراء PARALLEL SELECT:

لتكن لدينا المجموعة التالية:

S = {3,14,16,20,8,31,22,12,33,1,4,9,10,5,13,4,24,2,14,26,18,34,36,25,14,27,32,35,33}

فيها: n=29 .

ولتكن k=21 أي أننا نريد تحديد العنصر الواحد والعشرين في S.

لنفترض أن الحاسب EREW SM SIMD المتوفر يحتوي على خمس معالجات أي أن: N=5.
ومنه $|S|^{1-x}=5$ مما يدل على أن : $1-x = 0.47796$

يتم إدخال المتسلسلة مبدئياً في الذاكرة كما يظهر في الشكل (3-8):

S	3	14	16	20	8	31	22	12	33	1	4	9	10	5	13	7	24	2	14	26	18	34	36	25	14	27	32	35	33

بعد الخطوة 1 يكون كل معالج قد حدد المتسلسلة الجزئية في S حيث يتلقى كل من المعالجات الأربعة الأولى ستة عناصر من المجموعة والمعالج الخامس يأخذ العناصر الخمسة المتبقية كما يلي:

{3,14,16,20,8,31}

{22,12,33,1,4,9}

{10,5,13,7,24,2}

{14,26,18,34,36,25}

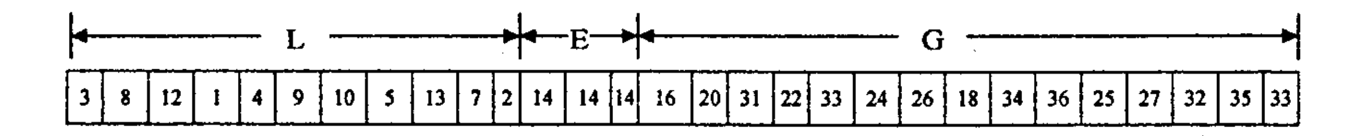
{14,27,32,35,33}

يقوم الآن كل معالج بإيجاد العنصر الأوسط في متسلسلته الجزئية في الخطوة 2 ويقوم بتخزينه في مصفوفة M في الذاكرة المقسمة وهذا موضح في الشكل التالي:

14	9	7	25	32

عندما يستدعى الإجراء PARALLEL SELECT تعاودياً في الخطوة 3 فإنه يعيد العنصر الأوسط m=14 في M

في الخطوة 4 يتم تشكيل المتسلسلات الجزئية الثلاثة لـ S المسماة G,E,L من العناصر الأصغر من 14 والمساوية لـ 14 والأكبر من 14 على التوالي وهذا موضح في الشكل التالي:



3	8	12	1	4	9	10	5	13	7	2	14	14	14	16	20	31	22	33	24	26	18	34	36	25	27	32	35	33

طالما أن $|L| + |E| < k$ ‏, $|E| = 3$ ‏, $|L| = 11$ فإن الإجراء PARALLEL SELECT يستدعى تعاودياً في الخطوة 5 حيث $S = G$ و $k = 21 - (11 + 3) = 7$

طالما أن $|G|=15$ فإننا نستعمل $15^{1-x}=3.6485$ حيث أن ثلاث معالجات تقف خلال هذا التعاود ومرة أخرى في الخطوة 1 يقوم كل معالج بتحديد خمس عناصر كما يظهر في الشكل التالي:

16	20	31	22	33

24	26	18	34	36

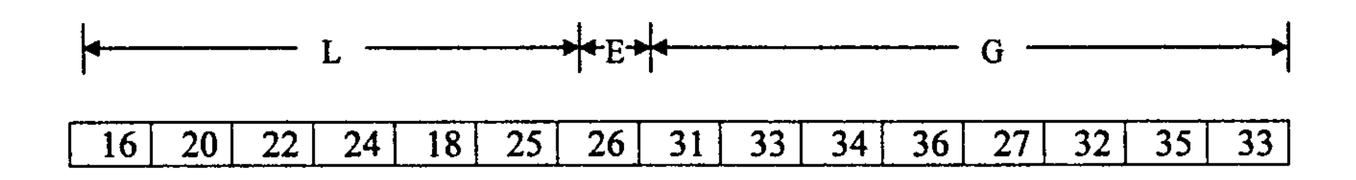
25	27	32	35	33

أما المتسلسلة M التي تحدد في الخطوة 2 مبينة في الشكل:

22	26	32

العنصر الأوسط $m=26$ يحدد في الخطوة 3

المتسلسلات الجزئية الثلاث L,E,G المنشأة في الخطوة 4 مبينة في الشكل:



الشكل (3-8) تطبيق عمل الإجراء PARALLEL SELECT

بما أن $|E|=1$, $|L|=6$ فإن العنصر الوحيد في E وهو 26 هو الذي يعود على أنه العنصر الواحد والعشرين في الإدخال وهو المطلوب.

ومنه نجد أننا حصلنا بنجاح على خوارزمية تنفذ على النموذج EREW SM SIMD وهذه الخوارزمية أسرع ومتكيفة مع عدد المعالجات وزمن التنفيذ ولها كلفة أمثلية وبالتالي الخوارزمية المتوازية تزودنا بأفكار تقود إلى تطوير الخوارزمية التسلسلية الأفضل الموجودة.

3-5-2 مسألة الدمج:

وهي أحد مسائل المقارنة ويمكن أن نعرفها كما يلي: لتكن $A=\{a_1,a_2,..,a_s\}$, $B=\{b_1,b_2,..,b_r\}$ متسلسلتين من الأرقام المرتبة تصاعدياً والمطلوب دمج كلا المتسلسلتين A , B وذلك لتشكيل متسلسلة ثالثة C: $C = \{c_1,c_2,..,c_{r+s}\}$ مرتبة تصاعدياً. حيث أن كل من العناصر c_i في C تنتمي إما إلى A أو إلى B وكل من العناصر a_i و b_i تظهر مرة واحدة فقط في C.

3-5-2-1 شبكة الدمج NETWWORK FOR MERGING:

يمكننا أن نحصل على شبكة غرض خاص لإنشاء التوازي بأي من الطرق التالية:

1- استعمال معالجات متخصصة مع شبكة الاتصالات التقليدية.

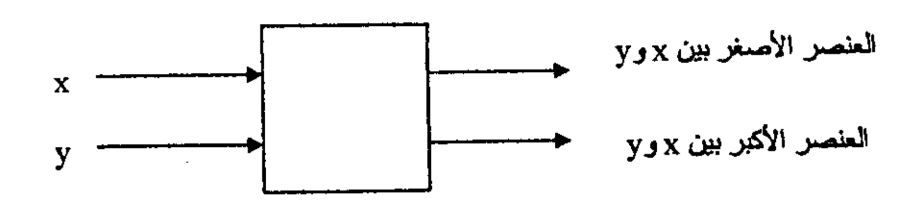
2- استعمال شبكة اتصالات مصممة بشكل عادي لربط معالجات قياسية.

3- استعمال خليط من 1 و 2.

سنأخذ في دراستنا لهذه المسألة الحالة الثالثة حيث سيتم الدمج عن طريق مجموعة من المعالجات البسيطة جداً متصلة بواسطة شبكة غرض خاص وهو البناء المتوازي وتعرف بـــ:

شبكة الدمج (r,s) /(r,s)-merging network/

إن جميع المعالجات المستعملة متماثلة وتدعى المقارنات comparators. يتلقى المقارن إدخالين ويصدر إخراجين كما يوضح الشكل (3-9).



الشكل (3-9) يوضح دخل وخرج المعالج المقارن

العملية الوحيدة التي يستطيع المقارن إنجازها هي مقارنة قيم الإدخالين الخاصين به ومن ثم وضع الأصغر منهما في قمة خطوط الخرج والأكبر منهما في أسفلها.

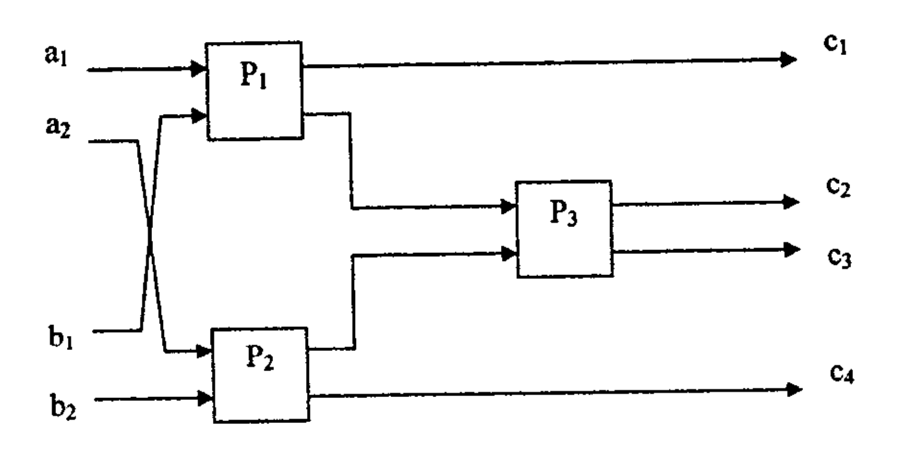
سنبدأ باستعمال تلك المقارنات لبناء شبكة تأخذ دخلها المتسلسلتين المرتبتين التاليتين: $A=\{a_1,a_2,..,a_s\}$ و $B=\{b_1,b_2,..,b_r\}$ وتنتج خرجها المتسلسلة المرتبة الوحيدة: $C=\{c_1,c_2,..,c_{r+s}\}$.

سنفترض للتوضيح ما يلي:

1- المتسلسلتين المدخلتين لهما نفس الحجم : $r = s = n \geq 1$
2- N هي قوة لـ 2

سنوضح الآن كيفية إنشاء شبكات الدمج من أجل القيم الثلاث الأولى لـ n.

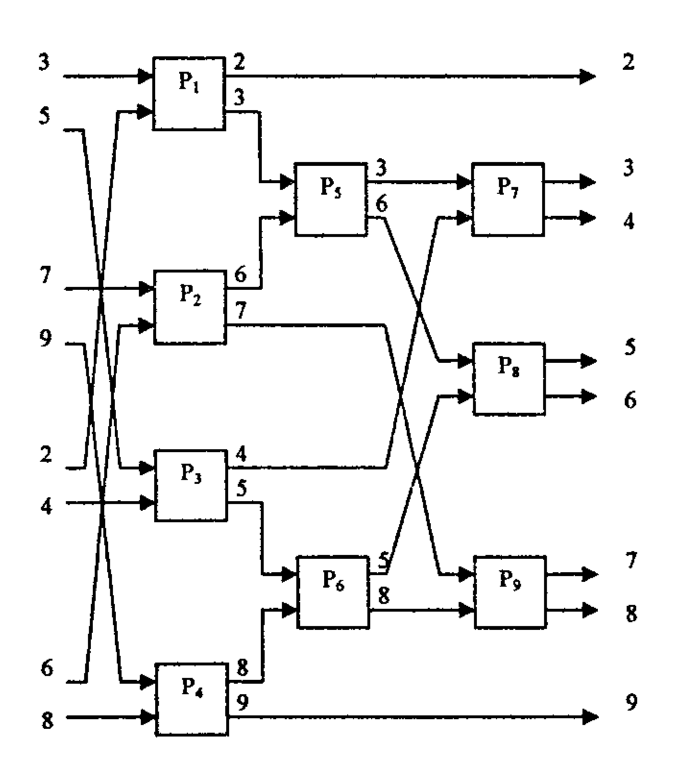
- عندما n=1: يكفي لإنشاء شبكة الدمج مقارن وحيد حيث ينتج إدخاليه بشكل مرتب.
- عندما n=2: المتسلسلتين $A=\{a_1,a_2\}$, $B=\{b_1,b_2\}$ يتم دمجها في الشبكة كما هو موضح في الشكل (3-10):



الشكل(3-10) يوضح شبكة خاصة لدمج متسلسلتين ثنائيتين

ويمكن إثبات هذا بسهولة : يقوم المعالج p_1 بمقارنة العنصر الأصغر في A مع العنصر الأصغر في B فتكون القيمة العليا في الخرج هي العنصر الأصغر في C والذي هو c_1. وبشكل مشابه القيمة السفلى في خرج المعالج p_2 يجب أن تكون c_4. بالإضافة على مقارنة أخرى تتم عن طريق p_3 الذي سينتج العنصرين الأوسطين في C.

- عندما n=4: نستطيع أن نستعمل شبكتين مماثلتين للشبكة السابقة يتبعها ثلاث مقارنات كما يظهر الشكل (3−11) من أجل : $A = \{3,5,7,9\}$, $B = \{2,4,6,8\}$



الشكل (3−11) شبكة دمج من أجل حجم المتسلسلتين n=4

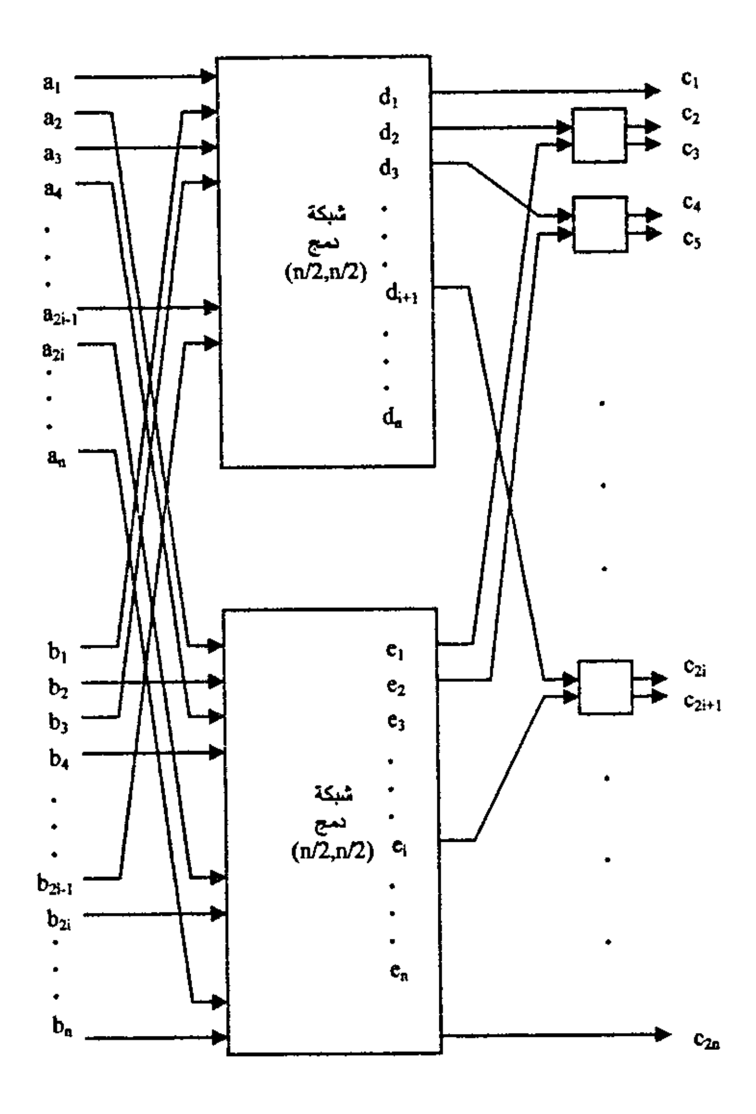
- بشكل عام يمكن بناء شبكة دمج (n,n) بالتعاود التالي:

أولاً تدمج العناصر ذات الأدلة الفردية في A,B وهذه العناصر هي: $\{a_1,a_3,a_5,..,a_{n-1}\}$ $\{b_1,b_3,b_5,..,b_{n-1}\}$ وذلك بواسطة شبكة دمج(n/2,n/2) لتنتج متسلسلة $\{d_1,d_2,d_3,..,d_n\}$. وبنفس

الوقت تدمج العناصر ذات الأدلة الزوجية والتي هي: $\{b_2,b_4,b_6,..,b_n\}$ $\{a_2,a_4,a_6,..,a_n\}$ باستعمال شبكة دمج (n/2,n/2) لتنتج متسلسلة $\{e_1,e_2,e_3,..,e_n\}$. ويمكن أن نحصل على المتسلسلة النهائية $\{c_1,c_2,c_3,..,c_{2n}\}$ كما يلي:

$$c_1 = d_1 \ , c_{2n} = e_n \ , c_{2i} = \min(d_{i+1}, e_i) \ , c_{2i+1} = \max(d_{i+1}, e_i) \ ; \quad i=1,2,..,n-1$$

أما المقارنات النهائية فهي n-1 مقارن كما هو موضح في الشكل (3-12).



الشكل (3-12) شبكة دمج (n,n)

نلاحظ أن كل من شبكتي الدمج (n/2,n/2) تبنى بتطبيق نفس قاعدة التعاود وذلك باستعمال شبكات دمج (n/4,n/4) يتبعها رتبة مقارنات 1-(n/2). وقد أنشأت شبكة الدمج في الشكل (3-12) بناء على نظرية معروفة الدمج (زوج ، فرد) ومنه فهذه النظرية تعمل بشكل عام كما يلي:

$$e_n = \max(a_n, b_n) \ , d_1 = \min(a_1, b_1)$$

وهذا يعني أن c_1 و c_{2n} حسبت تماماً.

نلاحظ أنه في المتسلسلة $\{d_1,d_2,d_3,..,d_n\}$ جميع العناصر d_i أصغر أو تساوي لـ d_{i+1} وكل عنصر من هذه المجموعة هو أحد عناصر A أو B ذات الأدلة الفردية لذلك فإن العناصر 2i من A أو B هي أصغر أو تساوي d_{i+1} ومنه: $d_{i+1} \geq c_{2i}$. وبشكل مشابه $e_i \geq c_{2i}$.

ومن جهة أخرى بالنسبة للمتسلسلة $\{c_1,c_2,c_3,..,c_{2n}\}$ فإن العناصر 2i من A أو B أصغر أو تساوي c_{2i+1} وهذا يعني أن c_{2i+1} أكبر أو تساوي العناصر i+1 ذات الأدلة الفردية التي تنتمي إما إلى A أو B , ومنه: $c_{2i+1} \geq d_{i+1}$. وبشكل مشابه فإن $c_{2i+1} \geq e_i$ ، ومنه: $c_{2i} \leq c_{2i+1}$ وهذا التباين الناتج يدل ضمناً على أن :

$$c_{2i+1} = \max (d_{i+1}, e_i) , c_{2i} = \min(d_{i+1}, e_i)$$

وهو المطلوب.

3-5-2-2 خواص شبكة الدمج (زوج ، فرد):

سنركز في تحليلنا للدمج (زوج ، فرد) على الزمن وعدد المعالجات ونتيجة عدد العمليات المطلوبة للدمج.

1- زمن التنفيذ:

في البداية سنفترض أنه يمكن للمقارن أن يقرأ إدخاله وينجز المقارنة ويصدر خرجه وذلك كله في وحدة زمن واحدة. ولنرمز للزمن المطلوب من شبكة دمج (n,n) (لدمج متسلسلتين كل منهما بطول n) بـ $t(2n)$.

تخضع هذه الشبكة للتعاود التالي بالنسبة للزمن $t(2n)$:

$$t(2) = 1 \qquad ; n = 1$$
$$t(2n) = t(n) +1 \qquad ; n > 1$$

ومنه يكون: $t(2n) = 1+\log n$ وهو أسرع من زمن التنفيذ $O(n)$ المحقق على حاسب تسلسلي.

2- عدد المعالجات:

لنحسب عدد المقارنات المطلوبة للدمج (زوج ، فرد) ، ولنرمز بـ $p(2n)$ لعدد المقارنات في شبكة دمج (n,n). ومرة أخرى لدينا التعاودية التالية:

$$p(2) = 1 \qquad ; n = 1$$

$$p(2n) = 2p(n) + (n-1) \qquad ; n > 1$$

ومنه: $p(2n) = 1+n \log n$ وهو بسيط أيضاً.

3- الكلفة:

طالما أن: $t(2n) = 1 + \log n$ و $p(2n) = 1 + n \log n$ فإن كلفة الشبكة هي:
$c(2n) = p(2n) \times t(2n) = O(n \log^2 n)$ ومنه فإن كلفة الشبكة ليست أفضلية فهي تنجز عمليات أكثر من $O(n)$ الكافية للدمج التسلسلي.

قمنا بعرض مثال عن بناء شبكة الدمج ، وشبكات الدمج هذه لها خاصية هامة هي أن سلسلة المقارنات التي ستنجزها ثابتة مسبقاً، بغض النظر عن الدخل ، ولذلك تعرف بأنها شبكات كثيرة النسيان لإدخالاتها. وقد رأينا في التحليل أن شبكة الدمج (n,n) المدروسة أسرع بكثير بالمقارنة مع خوارزمية الدمج التسلسلية المحتملة الأفضل ، وكمثال يمكن دمج متسلسلتين كل منهما بطول 2^{20} عنصر في واحد وعشرين خطوة . وتتطلب نفس النتيجة أكثر من مليوني خطوة على حاسب تسلسلي. ولسوء الحظ فقد تم تحقيق هذه النتيجة ولكن باستعمال عدد كبير وغير معقول من المعالجات ، فمن أجل $n=2^{20}$ ستتألف شبكة دمج (n,n) من أكثر من عشرين مليون مقارن بالإضافة إلى أن بناء الشبكة غير منتظم بشكل بالغ والأسلاك التي تصل المقارنات لها طول متغير بـــ n ، وهذا يعني أن شبكة الدمج ستكون غير عملية من أجل قيم كبيرة لـــ n.

3-5-2-3 الدمج على النموذج CREW:

مسألة الدمج (زوج ، فرد) تعتمد على الشبكات الجامدة أي التي تجمع عدد ثابت من المقارنات في تشكيلة لدمج متسلسلات بحجم ثابت. وبالرغم من أن ذلك كاف في بعض التطبيقات فإنه من المستحسن الحصول على خوارزمية متوازية تتكيف مع عدد المعالجات المتوفرة في الحاسب المتوازي المتوفر. سنصف في هذا المقطع إحدى هذه الخوارزميات ومع كونها أسرع فإن لها كلفة أفضلية أيضاً حيث يضرب زمن التنفيذ بعدد المعالجات المستخدمة ويساوي عامل ضرب ثابت وهو الحد الأدنى لعدد العمليات المطلوبة للدمج.

تعمل الخوارزمية على حاسب من النموذج CREW SM SIMD، وهي تفترض وجود الإجراء التسلسلي لدمج متسلسلتين مرتبتين ولذلك سنبدأ بإيجاد ذلك الإجراء.

1- إجراء الدمج التسلسلي SEQUENTIAL MERGING:

ليكن لدينا : $A = \{a_1,a_2,..,a_s\}$, $B = \{b_1,b_2,..,b_r\}$ متسلسلتين من الأرقام المرتبة تصاعدياً والمطلوب دمج A , B لتشكيل متسلسلة ثالثة C : $C = \{c_1,c_2,..,c_{r+s}\}$ مرتبة تصاعدياً.

ستتم عملية الدمج هذه بواسطة معالج وحيد ، ويمكن أن يتحقق ذلك عن طريق الخوارزمية التالية: تقوم المؤشرات بداية بالتأشير على العناصر a_1 و b_1 على التوالي . ومن ثم ينتسب الأصغر بينهما إلى c_1 ومؤشر المتسلسلة الذي يؤشر على c_1 يتقدم موضع واحد . مرة أخرى تتم مقارنة العنصرين المؤشر عليهما ويصبح الأصغر c_2 وبعدها يتقدم المؤشر. وتستمر هذه العملية حتى تنتهي إحدى المتسلسلتين المدخلتين وتنسخ العناصر المتبقية في المتسلسلة الأخرى في C.

2- خوارزمية SEQUENTIAL MERGE (A,B,C) :

- الخطوة 1:

1- $i \leftarrow 1$

2- $j \leftarrow 1$

- الخطوة 2:

من أجل k=1 إلى r+s نفذ :

إذا كان $a_i < b_j$ عندها: $c_k \leftarrow a_i$

$i \leftarrow i+1$

وإلا : $c_k \leftarrow b_j$

$j \leftarrow j+1$

```
Void SEQUENTIAL MERGE (A,B,C)
{
step1:      (1.1) i←1
            (1.2) j←1
step2:      for (k=1 , k <= r+s, k++)
                 if (ai < bj)
              (i)  ck←ai
             (ii)  i←i+1
                 else
               (i)    ck←bj
               (ii)   j←j+1
}
```

نلاحظ أنه بما أن كل مقارنة تعطي واحد من عناصر C فإنه يوجد r + s مقارنة وفي أسوأ الحالات عندما r = s = n نقول أن الخوارزمية تنفذ في زمن O(n) وكما رأينا عند عرض الحد الأدنى للدمج فإن الإجراء SEQUENTIAL MERGING أفضلي.

3- الدمج المتوازي PARALLEL MERGING:

يتألف الحاسب CREW SM SIMD من N معالج $P_1,P_2,P_3,..,P_N$ والمطلوب تصميم خوارزمية متوازية لهذا الحاسب تأخذ المتسلسلتين A و B كدخل وتنتج المتسلسلة C كخرج.

لنفرض أن r < s ومن المفترض أن الخوارزمية تحقق الخواص التالية:

1- يجب أن يكون عدد المعالجات المستخدمة في الخوارزمية sublinear و adaptive متكيف.
2- يجب أن يكون زمن تنفيذ الخوارزمية متكيف وأصغر من الخوارزمية التسلسلية الأفضل.
3- يجب أن تكون الكلفة أفضلية.

هذه الخوارزمية تستخدم N معالج حيث N≤r وفي أسوأ الأحوال عندما r = s = n فإن زمن تنفيذها يكون O((n/N)+log n) لذلك فإن الخوارزمية أفضلية من أجل N ≤ n/log n .

بالإضافة إلى توفر الدالات المنطقية والبناء الأساسي لكل من الـــ N معالج فإنه يفترض أيضاً قدرتها على إنجاز الإجراءين التاليين:

1- الإجراء SEQUENTIAL MERGING
2- الإجراء BINARY SEARCH الذي نشرحه فيما يلي:

يأخذ هذا الإجراء المتسلسلة $S=\{s_1,s_2,s_3,..,s_n\}$ من الأعداد المرتبة بشكل متزايد بالإضافة إلى متحول x.

إذا كانت x تنتمي إلى S عندها يعيد الإجراء الدليل k للعنصر s_k في S كالتالي: $x = s_k$. وإلا يعيد الإجراء قيمة الصفر. يعتمد الإجراء BINARY SEARCH على أسلوب فرق تسد حيث أنه في كل مرحلة تتم مقارنة بين x و عنصر من S فإما أن العنصرين متساويان وينتهي الإجراء أو أن نصف

عناصر المتسلسلة التي قيد البحث والمناقشة تطرح جانباً. تستمر هذه العملية حتى يصبح عدد العناصر المتبقية 0 أو 1. وبعد مقارنة واحدة على الأكثر ينتهي الإجراء.

4- خوارزمية BINARY SEARCH (S , x , k):

- الخطوة 1 :

1- $i \leftarrow 1$
2- $h \leftarrow n$
3- $k \leftarrow 0$

- الخطوة 2:

طالما أن $i \leq h$ نفذ:

1- $m \leftarrow (i+h)/2$
2- إذا كان $x = s_m$, عندها
   .1 $k \leftarrow m$
   .2 $i \leftarrow h+1$
و إلا إذا كان $x < s_m$ عندها $h \leftarrow m-1$
و إلا $i \leftarrow m+1$

Void BINARY SEARCH (S , x , k)

```
{
step1:      (1.1) i ←1;
            (1.2) h ←n;
            (1.3) k ←0;
step2:      while (i <= h)
            {
                  (2.1) m ← ⌊(i+h)/2⌋;
                  (2.2) if  (x = sm)
                              (i) k ←  m ;
                              (ii) i ←  h+1 ;
                        else if (x < sm)
                              h ←m-1 ;
                        else
                              i ←m+1  ;
            }
}
```

طالما أن عدد العناصر التي قيد المناقشة يختصر في كل مرة إلى النصف فإن الإجراء يتطلب زمن $O(\log n)$وذلك في أسوأ الأحوال.

سنشرح الآن أول خوارزمية دمج متوازية من أجل حاسب بذاكرة مشتركة:

5- خوارزمية CREW MERGE (A , B , C) :

أولاً سنحدد N-1 عنصر من A حيث سنقسم تلك المتسلسلة إلى N متسلسلة جزئية لها نفس الطول بشكل تقريبي ولتكن المتسلسلة الجزئية المشكلة من تلك الـ N-1 عنصر هي 'A ونختار المتسلسلة الجزئية 'B من N-1 عنصر من B بشكل مماثل.

ثانياً ندمج 'A و 'B في متسلسلة من التكرارات $V=\{v_1,v_2,v_3,..,v_{2N-2}\}$ حيث يتألف كل تكرار من عنصر من 'A أو 'B يتبعه موضعه في A أو B ويتبعه اسم المتسلسلة التي ينتمي إليها أصلاً والتي هي A أو B .

ثالثاً يدمج كل معالج ويدخل في C عناصر المتسلسلتين الجزئيتين واحد من A و واحد من B. يكون الفهرس للعنصرين واحد من A وواحد من B لكل معالج لكي يبدأ الدمج أولاً ثم ليحسب ويرتب في مصفوفة من الأزواج المرتبة.

الخطوة 1: [حدد N-1 عنصر من A حيث سنقسم تلك المتسلسلة إلى N متسلسلة جزئية لها نفس الطول بشكل تقريبي ولتكن المتسلسلة الجزئية المشكلة من تلك الـ N-1 عنصر هي 'A .ونختار المتسلسلة الجزئية 'B من N-1 عنصر من B بشكل مماثل. تنفذ هذه الخطوة كما يلي:]

من أجل i=1 إلى N-1 نفذ على التوازي:

المعالج p_i يحدد a'_i و b'_i من: .1 $a'_i \leftarrow a_{i\,r/n}$

.2 $b'_i \leftarrow b_{i\,r/n}$

الخطوة 2: [ادمج 'A و 'B في متسلسلة من التكرارات $V=\{v_1,v_2,v_3,..,v_{2N-2}\}$ حيث يتألف كل تكرار من عنصر من 'A أو 'B يتبعه موضعه في A أو Bويتبعه اسم المتسلسلة التي ينتمي إليها أصلاً والتي هي A أو B، ويتم التنفيذ كما يلي:]

1- من أجل i=1 إلى N-1 نفذ على التوازي:

أ- المعالج p_i يستعمل الإجراء BINARY SEARCH على 'B لكي يجد العنصر الأصغر j مثل $a'_i < b'_j$

ب- إذا كان j موجود عندها $v_{i+j-1} \leftarrow (a'_i,i,A)$

وإلا $v_{i+N-1} \leftarrow (a'_i,i,A)$

2- من أجل $i=1$ إلى $N-1$ نفذ على التوازي:

أ- المعالج p_i يستعمل الإجراء BINARY SEARCH على 'A لكي يجد العنصر الأصغر j كما يلي: $b'_i < a'_j$

ب- إذا كان j موجود عندها $v_{i+j-1} \leftarrow (b'_i,i,B)$

وإلا $v_{i+N-1} \leftarrow (b'_i,i,B)$

الخطوة 3: [يدمج كل معالج ويدخل في C عناصر المتسلسلتين الجزئيتين واحد من A و واحد من B. يكون الفهرس للعنصرين واحد من A وواحد من B لكل معالج لكي يبدأ الدمج أولاً ثم ليحسب ويرتب في مصفوفة من الأزواج المرتبة. تنفذ هذه الخطوة كالتالي:]

1- $Q(1) \leftarrow (1,1)$

2- من أجل $i=2$ إلى N نفذ على التوازي:

إذا كان $v_{2i-2}=(a'_k,k,A)$ عندها المعالج p_i :

(1) يستعمل الإجراء BINARY SEARCH على B لكي يجد العنصر الأصغر j كما يلي: $b_j > a'_k$

(2) $Q(i) \leftarrow (k\lceil r/n \rceil, j)$

وإلا المعالج p_i :

(3) يستعمل الإجراء BINARY SEARCH على A لكي يجد العنصر الأصغر j كما يلي: $a_j > b'_k$

(4) $Q(i) \leftarrow (j, k\lceil s/n \rceil)$

3- من أجل $i=1$ إلى N نفذ على التوازي:

المعالج p_i يستعمل الإجراء SEQUENTIAL MERGING و $Q(i) = (x,y)$ لدمج متسلسلتين جزئيتين واحدة تبدأ بـ a_x والأخرى بـ b_y وتوضع نتيجة الدمج في مصفوفة C تبدأ في الموضع $x+y-1$ ويستمر الدمج حتى:

1- يواجه عنصر أكبر أو يساوي لأول عنصر في v_{2i} في كل من A و B (عندما $i \le N-1$).

2- لا يوجد أي عنصر متبقي في A أو B (عندما $i = N$).

Void CREW MERGE (A , B , C)

{

```
Step1:      for (i=1, i < N, i++) in parallel

                    Processor p_i determines a'_i and b'_i from

                    (1.1) a'_i <- a_{i r/n}

                    (1.2) b'_i <- b_{i r/n}

Step2:      (2.1) for (i=1 , i < N , i++) in parallel
                    {
                    (i)    processor p_i uses BINARY SEARCH on B' to find the
                           smallest j such that a'_i < b'_j ;
                    (ii)   if (j exists)
                                  v_{i+j-1} <- (a'_i,i,A) ;
                           else
                                  v_{i+N-1} <- (a'_i,i,A) ;
                    }
            (2.2) for (i=1 , i < N , i++) in parallel
                    {
                           (i)    processor p_i uses BINARY SEARCH on A' to
                                  find the smallest j such that b'_i < a'_j ;
                           (ii)   if (j exists)
                                         v_{i+j-1} <- (b'_i,i,B) ;
                                  else
                                         v_{i+N-1} <- (b'_i,i,B) ;
                    }
Step3:      (3.1) Q(1) <- (1,1)
            (3.2) for (i=2 , i <= N , i++) in parallel
                           if  (v_{2i-2}=(a'_k,k,A) ) processor p_i
                           {
      (i)    uses BINARY SEARCH on B to find the smallest j such that b_j
             > a'_k ;
      (ii)   Q(i) <- (k ⌈r/n⌉ , j ) ;
                           }
                           else processor p_i
                           {
      (i)    uses BINARY SEARCH on A to find the smallest j such that a_j
             > b'_k ;
      (ii)   Q(i) <- (j,k ⌈ s/n⌉ ) ;
                           }
             (3.3) for (i=2 , i <= N , i++) in parallel
                           processor p_i uses SEQUENTIAL MERGING and Q(i)
                           = (x,y) to merge two subsequences one beginning at a_x
```

and the other at b_y and places the result of the merge on array C beginning at position x+y-1 . the merge continues until

(i) an element larger than or equal to the first component of v_{2i} is encountered in each of A and B (when $i \leq N-1$)

(ii) no elements are left in either A or B (when i = N) ;

6- ملاحظات:

1- في الأمثلة العامة إذا تمت مقارنة عنصر a_i من A مع عنصر b_j من B لتحديد أيهما الأصغر وكانت النتيجة أن $a_i=b_j$ عندها يكون قرار الخوارزمية بشكل حكمي أن a_i هي الأصغر.

2- تتم عمليات القراءة المتداخلة عندما يستخدم الإجراء BINARY SEARCH بالإضافة إلى أنه كل من هذه المعالجات تنفذ البحث الثنائي على نفس المتسلسلة.

7- تحليل خوارزمية CREW MERGE (A , B , C) :

سنحلل الإجراء CREW MERGE خطوة بخطوة كما يلي:

الخطوة 1 : يقوم كل معالج من المعالجات التي تعمل على التوازي بحساب متسلسلتين جزئيتين لذلك فإن هذه الخطوة تتطلب زمن ثابت.

الخطوة 2 : هذه الخطوة تتألف من تطبيقين للإجراء BINARY SEARCH لمتسلسلة بطول N-1 كل منها متبوع بعبارة محددة وهذا يأخذ زمن O(log N) .

الخطوة 3 : الخطوة 3-1 تتألف من زمن محدد. والخطوة 3-2 تتطلب على الأغلب زمناً قدره O(log s). لتحليل الخطوة 3-3 نلاحظ أولاً أن V تحتوي على 2N-2 عنصر حيث تقسم C إلى 2N-1 متسلسلة جزئية مع حجم أعظمي يساوي إلى ($\lceil r/n \rceil + \lceil s/n \rceil$) وهذا الحجم الأعظمي يحدث إذا كان مثلا عنصر ما a_i من A يساوي عنصر ما b_j من B عندها العناصر $\lceil r/n \rceil$ أصغر أو تساوي a_i (وأكبر أو تساوي a_{i-1}) وهي أيضاً أصغر أو تساوي b_j. وبشكل مشابه العناصر $\lceil s/n \rceil$ التي هي أصغر أو تساوي b_j (وأكبر أو تساوي b_{j-1})هي أيضاً أصغر أو تساوي a_i.

في الخطوة 3 يقوم كل معالج بإنشاء اثنتين من المتسلسلات الجزئية من C ولذلك فإن حجمها ليس أكبر من $2(\lceil r/n \rceil + \lceil s/n \rceil)$ ما عدا p_N الذي ينشئ متسلسلة جزئية واحدة من C وهذا يعني أن الإجراء SEQUENTIAL MERGING يأخذ على الأغلب زمن $O((r+s) / N)$ وفي أسوأ الحالات عندما $r = s = n$ وطالما أن $n \geq N$ فإن زمن تنفيذ الخوارزمية يحدد عن طريق الزمن المطلوب في الخطوة 3 ولذلك:

$$t(2n) = O(n/N) + \log n$$

طالما أن $p(2n) = N$ منه نجد:

$$c(2n) = p(2n) \times t(2n) = O(n + N \log n)$$

وكلفة الخوارزمية أفضلية عندما $N \leq n/\log n$.

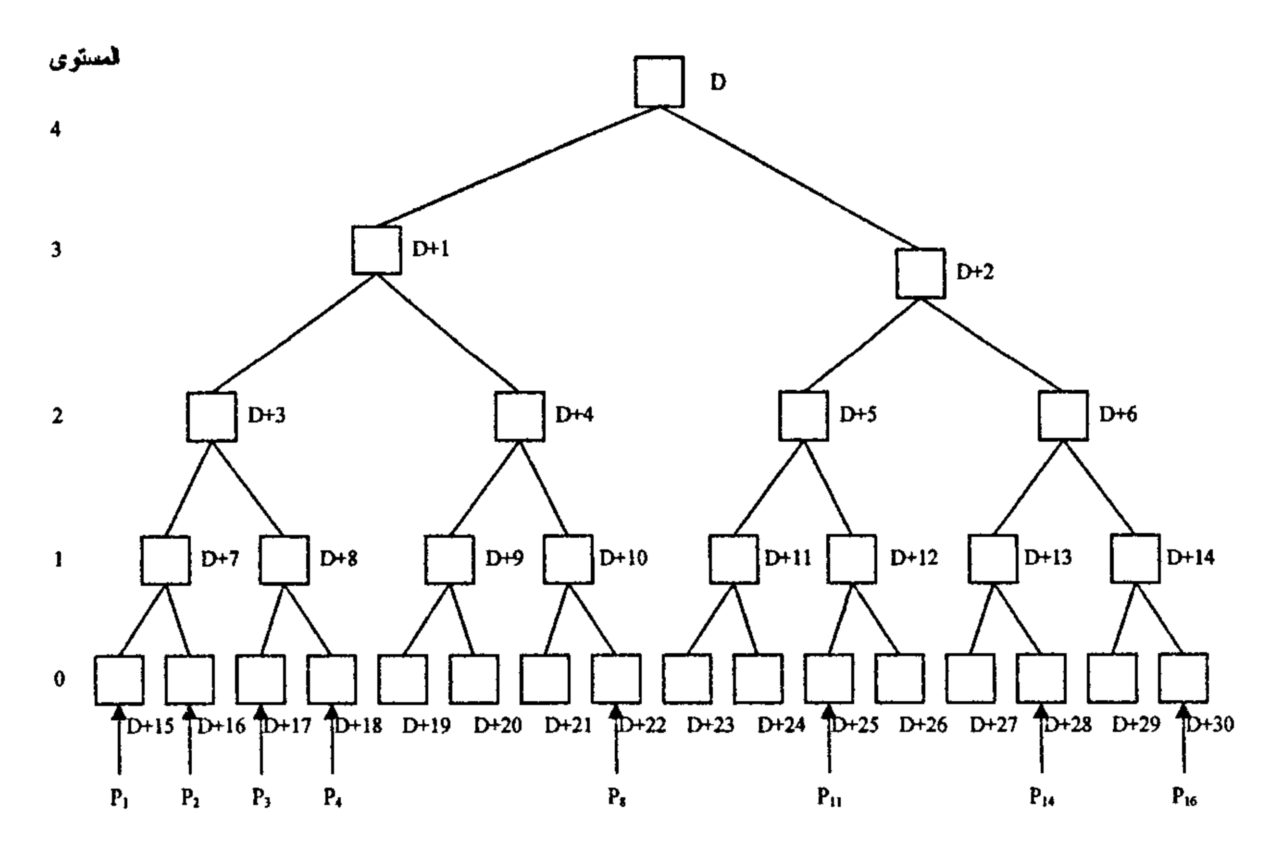
3-5-2-4 الدمج على النموذج EREW:

لقد رأينا في إجراء الدمج المتوازي على النموذج CREW أن عمليات القراءة المتداخلة تمت في العديد من خطواته وسنعمل لإيجاد إجراء يتكيف للعمل على N معالج في حاسب EREW SM SIMD وهذا النموذج من الحواسيب لا يسمح بأي محاولة للقراءة من أكثر من معالج واحد من موقع واحد في الذاكرة. ما علينا فعله هو إيجاد طريقة لمحاكاة عمليات القراءة المتعددة وحالما نجد مثل هذه المحاكاة يمكننا عندئذ أن نستخدم خوارزمية الدمج المتوازي (وبشكل عام أي خوارزمية فيها عمليات قراءة متعددة) لإتمام كل عمليات القراءة في ذاكرة النموذج EREW ، إذاً نحن بحاجة للمحاكاة لتصبح الخوارزمية فعالة.

نلاحظ أن تقديم جميع طلبات القراءة من موقع الذاكرة واحد تلو الآخر غير مكيف إذ أنه يمكن أن يزيد زمن التنفيذ بدالة بـــ N في أسوأ الأحوال. من ناحية أخرى فإن استخدام الإجراء BROADCAST السابق غير مناسب أيضاً إذ أن عمليات القراءة المتعددة من موقع الذاكرة ليس بالضرورة أن تستخدم جميع المعالجات إذ أن هناك عدة مجموعات فرعية تحكمية من مجموعة المعالجات تعتمد على الوصول المحدد لمواقع مختلفة ، موقع واحد لكل مجموعة جزئية. لذلك سنقوم بإيجاد طريقة لإنجاز التحاكي بواسطة الإجراء MULTIPLE BROADCAST التالي:

بفرض أن لدينا خوارزمية صممت من أجل حاسب CREW SM SIMD وتتطلب مجموعة من M موقع في الذاكرة المشتركة. من أجل محاكاة هذه الخوارزمية على نموذج EREW مع N معالج حيث

$N = 2^q$ و $q \geq 1$. سنقوم في البداية بزيادة حجم الذاكرة من M موقع إلى M(2N-1) فيصبح عندئذ كل من الـــ M موقع عبارة عن جذر لشجرة ثنائية مع N ورقة. وتملك مثل هذه الشجرة q+1 مستوى ومجموعة من 2N-1 عقدة كما يظهر في الشكل(3-13) من أجل N=16.



الشكل(3-13) تنظيم الذاكرة من أجل الإعلان المتعدد

تمثل عقد الشجرة مواقع متتالية في الذاكرة لذلك فإنه إذا كان الموقع D هو الجذر فإن خلفه اليساري واليميني هما D+1 و D+2 على التوالي. وبشكل عام الخلف اليساري واليميني لـــD+X هما D+2X+1 و D+2X+2 على التوالي.

لنفترض أن المعالج P_i يريد في حالة ما أن يقرأ من موقع ما d(i) في الذاكرة عندئذ سيضع طلبه في الموقع (i-1) + (N-1) + d(i) ورقة الشجرة التي جذرها d(i). يتم ذلك عن طريق متغيرين محليين مبدئيين لـــ P_i :

1- المستوى (i) الذي يخزن المستوى الحالي للشجرة الذي وصل إليه طلب P_i وهو مبدئياً 0.

2- الموقع Loc(i) الذي يخزن العقدة الحالية للشجرة التي وصل إليها طلب P_i وهو مبدئياً (i-1) + (N-1). نلاحظ أن P_i يحتاج فقط لتخزين الوضع في الشجرة المتصلة بـــ d(i) حيث وصل طلبها وليس موقع الذاكرة الفعلي (i-1) + (N-1) + d(i).

تتألف هذه المحاكاة من مرحلتين: مرحلة سابقة ومرحلة لاحقة. تقوم المعالجات خلال المرحلة السابقة بما يلي: يشغل المعالج P_i في كل مستوى الخلف اليساري وذلك حسب أولوية له من أجل تحسين طلبه مستوى واحد للأعلى في الشجرة، ويتم ذلك عن طريق ترميز موقع الأب بترميز خاص وليكن [i] عندها يقوم بتحديث مستواه وموقعه وفي هذه الحالة يجمد استدعاء الخلف اليميني من أجل الباقي من الإجراء وإلا (إذا لم يكن هناك معالج يشغل الخلف اليساري مثلاً) يمكن للمعالج الذي يشغل الخلف اليميني طلب موقع الأب وتستمر هذه العملية حتى يصل معالجين على الأغلب المستوى 1-(log N) حيث أن جميع المعالجات تقرأ القيمة المخزنة في الجذر تباعاً (على التعاقب) وتبدأ مرحلة الهبوط فتذهب القيمة التي قرأت حالاً إلى أسفل شجرة مواقع الذاكرة حتى تتم جميع طلبات القراءة من المعالجات. وبذلك تكون خوارزمية الإجراء MULTIPLE BROADCAST كالتالي:

1- خوارزمية MULTIPLE BROADCAST(d(1),d(2),..,d(N)) :

الخطوة 1: من أجل i=1 إلى N نفذ على التوازي:

[المعالج p_i يأخذ المستوى level(i) والموقع loc(i)]

1-1 ضع في المستوى I القيمة 0 $\qquad$ level(i)←0

1-2 ضع في الموقع iالقيمة N+i-2 $\qquad$ loc(i) ← N+i-2

1-3 خزن [i] في الموقع d(i)+loc(i)

الخطوة 2: من أجل v=0 إلى 2-(log N) نفذ:

2-1 من أجل i=1 إلى N نفذ على التوازي:

[المعالج p_i في الابن اليساري يتقدم إلى الأمام في شجرته]

2-1-1 ضع في x الحد الأعلى الأصغري (floor) للقيمة (loc(i)-1)/2

X← (loc(i)-1)/2

2-1-2 إذا كان loc(i) فردي و level(i)=v عندها:

(i) ضع x في loc(i)

(ii) خزن [i] في الموقع d(i)+loc(i)

(iii) ضع level(i)+1 في level(i)

2-2 من أجل i=1 إلى N نفذ على التوازي:

[المعالج p_i في الابن اليميني يتقدم إلى الأمام (للأعلى) في شجرته إذا كان ذلك ممكناً]

إذا كان d(i)+x لا يحتوي مسبقاً على الدليل [j] من أجل بعض 1≤j≤N عندها:

(i) ضع x في loc(i)

(ii) خزن [i] فيd(i)+loc(i)

(iii) ضع level(i)+1 في level(i)

الخطوة 3: من أجل v= (log N)-1 إلى 0 نفذ:

3-1 من أجل i=1 إلى N نفذ على التوازي:

في الابن اليساري يقرأ من والده وينتقل بعدها إلى أسفل الشجرة[p_i] المعالج

1-1-3 X← (loc(i)-1)/2

2-1-3 y← (loc(i)×2)+1

3-1-3 إذا كان loc(i) فردي وlevel(i)=0 عندها:

(i) اقرأ محتويات d(i)+x

(ii) اكتب محتويات d(i)+x في الموقع d(i)+loc(i)

(iii) ضع level(i)-1 مكان level(i)

(iv) إذا كان الموقع d(i)+y يحتوي [i] عندها: ضع y في loc(i)

وإلا: ضع y+1 في loc(i)

3-2 من أجل i=1 إلى N نفذ على التوازي:

[المعالج p_i في الابن اليميني يقرأ من والده وينتقل بعدها إلى أسفل الشجرة]

إذا كان loc(i) زوجي و level(i)=v عندها:

(i) اقرأ محتويات d(i)+x

(ii) اكتب محتويات d(i)+x في الموقع d(i)+loc(i)

(iii) ضع level(i)-1 مكان level(i)

(iv) إذا كان الموقع d(i)+y يحتوي [i] عندها: ضع y في loc(i)

وإلا: ضع y+1 في loc(i)

Void MULTIPLE BROADCAST(d(1),d(2),..,d(N))

```
{
Step1: for (i=1 , i <= N , i++) in parallel
// pi initializes level(i) and loc(i)
{
1.1 level(i) ← 0
1.2 loc(i) ← N+i-2
1.3 store [i] in location d(i)+loc(i)
}
```

```
Step2: for (v=0 , v <= (log N)-2 , v++)
{
        (2.1)  for (i=1 , i <= N , i++) in parallel
        // pi at a left child advnces up its tree
        {       (2.1.1) x← ⌊(loc(i)-1)/2⌋
                (2.1.2) if (loc(i) is odd and level(i)=v)
                                {       (i) loc(i) ← x
                                        (ii) store [i] in location d(i)+loc(i)
                                        (iii) level(i) ← level(i) +1
                                }
        }
        (2.2)  for (i=1 , i <= N , i++) in parallel
        // pi at a right child advances up its tree if possible
        if (d(i)+x does not already contain a marker [j] for some 1≤j≤N)
                {
                        (i) loc(i) ← x
                        (ii) store [i] in location d(i)+loc(i)
                        (iii) level(i) ← level(i) +1
                }
}
Step3: for (v= (log N )-1 , v >= 0 , v--)
{
        (3.1)  for (i=1 , i <= N , i++) in parallel
        // pi at a left child reads from its parent and then moves down the //tree
        {
        (3.1.1) x← ⌊(loc(i)-1)/2⌋
        (3.1.2) y← (2 ×loc(i)) + 1
        (3.1.3) if (loc(i) is odd and level(i)=v)
                {
                (i)     read the contents of d(i)+x
                (ii)    write the contents of d(i)+x in location d(i)+loc(i)
                (iii)   level (i) ← level (i) – 1
                (iv)    if (location d(i)+y contains [i])
                                loc(i) ← y
                        else
                                loc(i) ← y + 1
                }
        }
        (3.2)  for (i=1 , i <= N , i++) in parallel
        // pi at a right child reads from its parent and then moves down the //tree
        {
        if (loc(i) is even and level(i)=v)
```

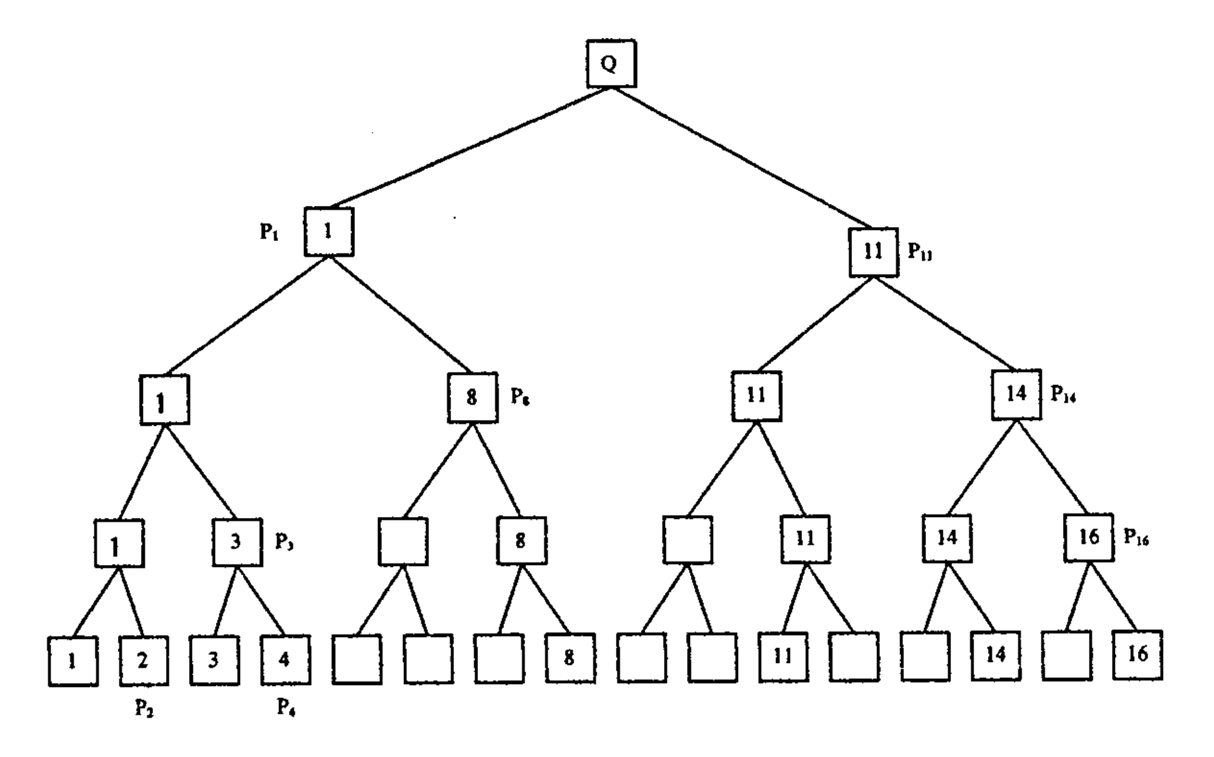
```
            {
            (v)     read the contents of d(i)+x
            (vi)    write the contents of d(i)+x in location d(i)+loc(i)
            (vii)   level (i) ← level (i) – 1
            (viii)  if (location d(i)+y contains [i])
                          loc(i) ← y
                    else
                          loc(i) ← y + 1
            }
      }
}
```

نلاحظ أن الخطوة (1) في الإجراء تتألف من ثلاث عمليات ثابتة مع الزمن، إضافة إلى أن كل من المرحلتين الصاعدة (المتزايدة) والنازلة (المتناقصة) في الخطوتين 2 و3 على التوالي تتطلب زمن O(log N) ومنه فإن زمن التنفيذ للإجراء MULTIPLE BROADCAST هو O(log N).

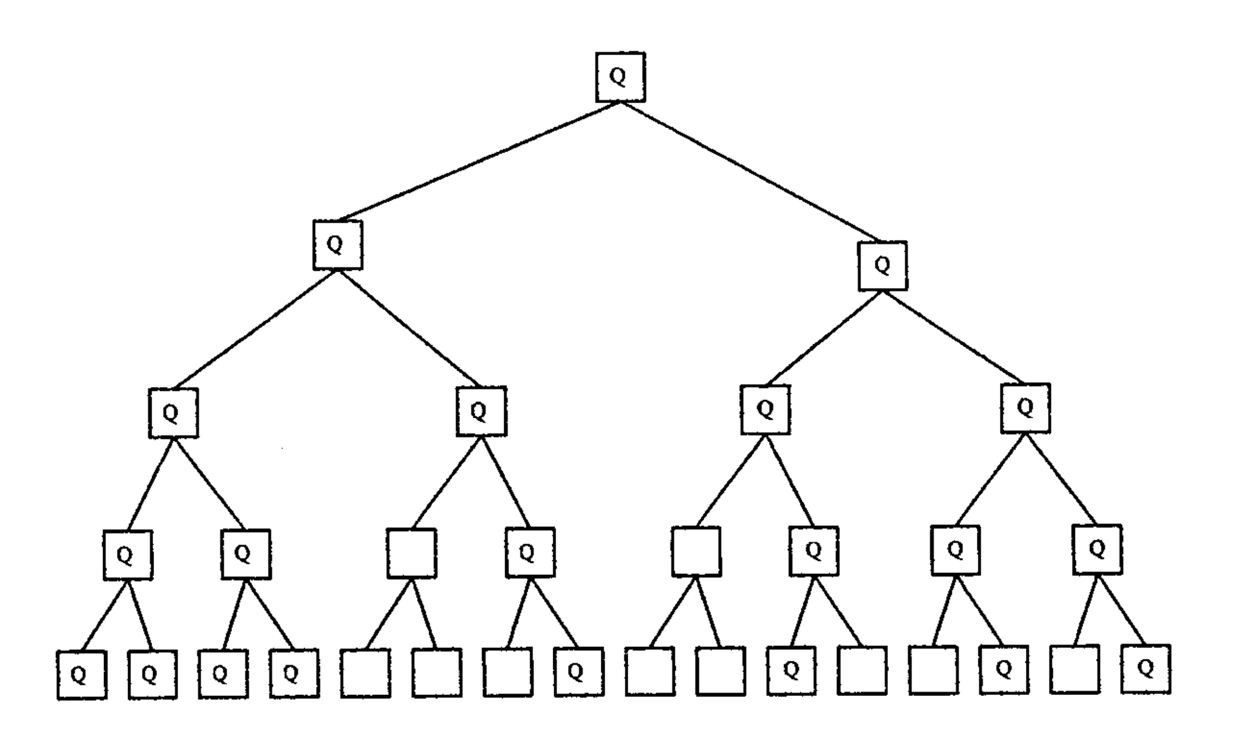
مثال(3-8):

لتكن N=16 ولنفرض في لحظة معطاة خلال التنفيذ على معالجات خوارزمية متوازية CREW أن المعالجات $P_1,P_2,P_3,P_4,P_8,P_{11},P_{14},P_{16}$ تحتاج لقراءة كمية Q من موقع D في الذاكرة.

عندما نحاكي عملية القراءة المتعددة هذه على حاسب EREW نستخدم MULTIPLE BROADCAST حيث تضع المعالجات طلباتها في مستويات معينة في شجرة مواقع جذرها D خلال الخطوة(1) كما نرى في الشكل 3-14 ، وتظهر المواضع للمعالجات المختلفة ومحتويات مواقع الذاكرة في نهاية الخطوة(2) في الشكل 3-14 ، ومحتويات مواقع الذاكرة في نهاية الخطوة(3) تظهر في الشكل 3-15.



الشكل (3-14) محتويات الذاكرة بعد الخطوة (2) في الإجراء MULTIPLE BROADCAST



الشكل (3-15) محتويات الذاكرة في نهاية الإجراء MULTIPLE BROADCAST

سنعرض خوارزمية متوازية ومتكيفة وبكلفة أفضلية للدمج على النموذج EREW SM SIMD. تقوم الخوارزمية بدمج متسلسلتين مرتبتين $A = \{a_1,a_2,..,a_s\}$ و $B=\{b_1,b_2,..,b_r\}$ في متسلسلة ثالثة: $C = \{c_1,c_2,..,c_{r+s}\}$ مرتبة تصاعدياً. وتستخدم N معالج: $p_1,p_2,..,p_N$ حيث: $1 \leq N \leq r+s$ وفي أسوأ الأحوال عندما: $r = s = n$ تنفذ الخوارزمية خلال زمن $O((n/N)+\log N \log n)$. ويتم بناء الخوارزمية عن طريق إجراء تسلسلي لإيجاد الأوسط في متسلسلتين مرتبتين.

3- خوارزمية إيجاد العنصر الأوسط لمتسلسلتين مرتبتين:

ليكن لدينا متسلسلتين مرتبتين $A = \{a_1,a_2,..,a_s\}$ و $B=\{b_1,b_2,..,b_r\}$ حيث: $r,s \geq 1$ ولتكن A.B متسلسلة طولها $m = r+s$ تنتج من دمج A و B ، فيكون العنصر الأوسط هو $m/2$ في A.B ، وتعيد هذه الخوارزمية زوج (a_x,b_y) يحقق الخواص التالية:

1- الأوسط في A.B هو إما a_x أو b_y حيث إما a_x أو b_y أكبر تماماً من العناصر $m/2$ -1 وأصغر تماماً من العناصر $\lfloor m/2 \rfloor$.

2- إذا كان a_x هو الأوسط عندها فإن b_y يكون إما:

(i) أكبر عنصر في B أصغر أو يساوي a_x

(ii) أصغر عنصر في B أكبر أو يساوي a_x

وبشكل مشابه: إذا كان b_y هو الأوسط عندها فإن a_x يكون إما:

(iii) أكبر عنصر في A أصغر أو يساوي b_y

(iv) أصغر عنصر في A أكبر أو يساوي b_y

3- إذا كان هناك أكثر من زوج واحد يحقق 1 و 2 عندها فإن الخوارزمية تعيد الزوج الذي من أجله $x + y$ هو الأصغر.

سنرمز بـــ(a_x,b_y) للزوج الأوسط في A.B لذلك فإن x و y هما دليلا الزوج الأوسط.

ونلاحظ أن a_x هو الأوسط في A.B إذا كان:

(i) إما $a_x > b_y$ و $x+y-1=\lceil m/2 \rceil -1$

(ii) أو $a_x < b_y$ و $m-(x+y-1)=\lfloor m/2 \rfloor$

وإلا فإن b_y هو الأوسط في A.B

مثال(3-9):

لتكن $A=\{2,5,7,10\}$ و $B=\{1,4,8,9\}$ ، نلاحظ أن الأوسط في A.B هو 5 وهو ينتمي إلى A.

يوجد لدينا زوجين أوسطين يحققان الخاصتين 1و2:

- $(a_2,b_2) = (5,4)$ حيث 4 هو أكبر عنصر في B أصغر أو يساوي 5
- $(a_2,b_3) = (5,8)$ حيث 8 هو أصغر عنصر في B أكبر أو يساوي 5

لذلك فإن الزوج الأوسط هو (5,4).

لدينا فيما يلي الإجراء Tow_ SEQUENCE MEDIAN الذي يتم في مراحل وبعد كل مرحلة تنقل في النهاية بعض العناصر من المناقشة من كل من A و B.

سنرمز بـــ n_A و n_B لعدد العناصر في A و B على التوالي التي مازالت قيد المناقشة في بداية كل مرحلة وبـــ w إلى الأصغر فيما بين $\lfloor n_A/2 \rfloor$ و $\lfloor n_B/2 \rfloor$.

وتتم كل مرحلة كالتالي: تتم مقارنة كل من الأوسط a و b من العناصر التي ما زالت قيد المناقشة في كل من A و B على التوالي ، إذا كان $a \geq b$ عندها تبعد العناصر الأكبر (الأصغر) في A (B) عن المناقشة. من جهة أخرى فإنه إذا كان $a < b$ فإن العناصر الأصغر (الأكبر) w في A (B) تبعد عن المناقشة. تتكرر هذه العملية حتى يبقى عنصر واحد فقط قيد المناقشة في متسلسلة واحدة أو في الاثنتين. عندها يحدد الأوسط من مجموعة صغيرة من الأزواج المرشحة.
يسلسل الإجراء هذه الأفكار عن العناصر التي ما زالت قيد المناقشة عن طريق استعمال مؤشرين لكل من المتسلسلتين: $high_A$, low_A في A و $high_B$, low_B في B .

4- خوارزمية الإجراء Tow_ SEQUENCE MEDIAN(A,B,x,y) :

Void Tow_ SEQUENCE MEDIAN(A,B,x,y)

Step1: (1.1) $low_A \leftarrow 1$

(1.2) $low_B \leftarrow 1$

(1.3) $high_A \leftarrow r$

(1.4) $high_B \leftarrow s$

(1.5) $n_A \leftarrow r$

(1.6) $n_B \leftarrow s$

Step2: while $((n_A>1)$ && $(n_B>1))$

{

(2.1) $u \leftarrow low_A + \lceil (high_A - low_A - 1)/2 \rceil$

(2.2) $v \leftarrow low_B + \lceil (high_B - low_B - 1)/2 \rceil$

(2.3) $w \leftarrow \min(\lfloor n_A/2 \rfloor, \lfloor n_B/2 \rfloor)$

(2.4) $n_A \leftarrow n_A - w$

(2.5) $n_B \leftarrow n_B - w$

(2.6) if $(a_u \geq b_v)$

{ (i) $high_A \leftarrow high_A - w$

(ii) $low_B \leftarrow low_B + w$

}

else

{ (i) $low_A \leftarrow low_A + w$

(ii) $high_B \leftarrow high_B - w$

}

}

Step3: return as x and y the idices of the pair from $\{a_{u-1}, a_u, a_{u+1}\} \times \{b_{v-1}, b_v, b_{v+1}\}$ satisfying properties 1-3 of median pair.

نلاحظ أن الإجراء Tow_ SEQUENCE MEDIAN يعيد أدلة الزوج الأوسط (a_x,b_y) بدلاً من الزوج نفسه.

مثال(3-9):

لتكن $A=\{2,5,7,10\}$ و $B=\{1,4,8,9\}$ ، نلاحظ أن الأوسط في A.B هو 5 وهو ينتمي إلى A.

يوجد لدينا زوجين أوسطين يحققان الخاصتين 1و2:

- $(a_2,b_2) = (5,4)$ حيث 4 هو أكبر عنصر في B أصغر أو يساوي 5
- $(a_2,b_3) = (5,8)$ حيث 8 هو أصغر عنصر في B أكبر أو يساوي 5

لذلك فإن الزوج الأوسط هو (5,4).

لدينا فيما يلي الإجراء Tow_ SEQUENCE MEDIAN الذي يتم في مراحل وبعد كل مرحلة تنقل في النهاية بعض العناصر من المناقشة من كل من A و B.

سنرمز بـــ n_A و n_B لعدد العناصر في A و B على التوالي التي مازالت قيد المناقشة في بداية كل مرحلة وبـــ w إلى الأصغر فيما بين $\lfloor n_A/2 \rfloor$ و $\lfloor n_B/2 \rfloor$.

وتتم كل مرحلة كالتالي: تتم مقارنة كل من الأوسط a و b من العناصر التي ما زالت قيد المناقشة في كل من A و B على التوالي ، إذا كان $a \geq b$ عندها تبعد العناصر الأكبر (الأصغر) في A (B) عن المناقشة. من جهة أخرى فإنه إذا كان $a < b$ فإن العناصر الأصغر (الأكبر) w في A (B) تبعد عن المناقشة. تتكرر هذه العملية حتى يبقى عنصر واحد فقط قيد المناقشة في متسلسلة واحدة أو في الاثنتين. عندها يحدد الأوسط من مجموعة صغيرة من الأزواج المرشحة.
يسلسل الإجراء هذه الأفكار عن العناصر التي ما زالت قيد المناقشة عن طريق استعمال مؤشرين لكل من المتسلسلتين: $high_A$, low_A في A و $high_B$, low_B في B .

4- خوارزمية الإجراء Tow_ SEQUENCE MEDIAN(A,B,x,y) :

Void Tow_ SEQUENCE MEDIAN(A,B,x,y)

Step1: (1.1) $low_A \leftarrow 1$

(1.2) $low_B \leftarrow 1$

(1.3) $high_A \leftarrow r$

(1.4) $high_B \leftarrow s$

5- تحليل خوارزمية الإجراء Tow_ SEQUENCE MEDIAN:

تأخذ الخطوتين 1 و 3 زمناً ثابتاً. كل تكرار في الخطوة 2 يجعل الأصغر من المتسلسلتين إلى النصف. من أجل الثوابت c_1,c_2 فإن الإجراء Tow_ SEQUENCE MEDIAN يتطلب زمن قدره $c_1+c_2 \log(\min\{r,s\})$ والذي هو $O(\log n)$ في أسوأ الأحوال.

النتائج والمقترحات

يصعب في بعض الأحيان كتابة خاتمة للأبحاث العلمية ولبعض الأبحاث النظرية التي تلخص أهم نتائج تلك الأبحاث. ونظراً لأن الرسالة تندرج ضمن الفئتين السابقتين فقد رأينا أنه من المفيد أن نختتم هذا العمل ببعض الملاحظات والنتائج والمقترحات.

النتائج:

- إن دراسة الخوارزميات المتوازية هذا الفرع الجديد من فروع المعلوماتية مسألة صعبة في وقتنا الراهن وخصوصاً لعدم توفر أجهزة ملائمة تسمح بترجمة الخوارزميات المقدمة في هذه الرسالة إلى برامج عملية نتحقق من خلال تطبيقها على مسائل واقعية من أهمية النتائج التي يمكن أن نتوصل إليها.
- بينّا في الفصل الأول من الرسالة مفهوم الأنظمة متعددة المعالجات وتعريفها وتصنيفها ودراسة أسلوب التواصل فيها ، كما قدمنا أفكاراً ونتائج تشكل أساسات نظرية الحساب المتوازي و وضحنا بعض نماذج الحساب المتوازي.
- درسنا في الفصل الثاني الخوارزميات المتوازية وأنواعها وميزنا خواصها وتقنيات تصميمها ومقاييس تحليلها وتحديد كفاءتها.
- قدمنا في الفصل الثالث مسائل هامة وخوارزميات حلها بشكل متوازي وذلك بالاعتماد على أجهزة متعددة المعالجات ، ووجدنا أننا نستطيع باستخدام الخوارزميات المتوازية زيادة حجم المسائل الممكن حلها كما تساعد الخوارزميات المتوازية في الحصول على معلومات أكثر وحلول أفضل

بتقليل الكلفة للمسائل كما وجدنا ضرورة تخصيص المعالجات بحجم كبير من العمل لاستثمار الحواسيب المتوازية بشكل أفضل.

- تسمح المسائل التي تمت معاجتها في هذا الفصل في توضيح أثر الموازاة في زيادة حجم المسائل التي يمكن حلها وتقليل زمن الوصول إلى الحل الأمثل إذ أن استخدام خوارزمية التحقق من الانتهاء يخبر بوصول مبكر للحل وبالتالي يؤدي إلى تقليل زمن التنفيذ.

المقترحات والتوصيات:

عندما يقرر أحد الباحثين معالجة مشكلة أو إجراء بحث علمي يكتشف في نهاية المطاف أنه مهما توسع في بحثه وحاول تغطية كافة جوانبه ومعالجتها تبقى بعض المسائل التي لم يمكن من معالجتها أو أنها ظهرت في سياق البحث وأصبحت خارج حدود المسار الذي رسمه لنفسه الأمر الذي يدفع بالباحث إلى تقديم مجموعة من المقترحات والتوصيات لتكون مدخلاً لأبحاث جديدة أو دعوة إلى الباحثين في المضمار نفسه لمعالجتها وإيجاد حلول لها. ومن هذا المنطلق أرى ضرورة تقديم بعض المقترحات والتوصيات:

❖ أقترح إجراء دراسة عامة وموسعة لكافة المسائل بالأسلوب المتوازي وفهم طرائق التعامل مع الأجهزة المتوازية وشبكات اتصالها لتحديد الخوارزميات المناسبة لحل المسائل.

❖ تبين لنا أن المسألة الملحة الواجب معالجتها من وجهة نظر البرمجيات تكمن في تخصيص المهمات أو العمليات للمعالجات وتطرح هذه المسألة نفسها في الأجهزة المتعددة المعالجات ولابد من إجراء دراسات متعمقة

في هذا المجال حتى نحصل على أفضل النتائج المأمولة من الموازاة في البرمجة.

- ❖ أقترح الاستفادة من مفاهيم وخواص الخوارزميات المتوازية لوضع خوارزميات جديدة أو تطوير الخوارزميات القديمة لحل المسائل التي لا نعرف حلاً عاماً لها كمسألة التاجر المسافر وغيرها.
- ❖ أرى أن مشكلة نقل المعلومات التي ظهرت في مسألة البحث الشجري وتظهر في العديد من المسائل المشابهة ستبقى لمدة طويلة موضوع الساعة ومن المناسب التعمق في دراستها والبحث عن خوارزميات تسمح بمعالجتها وتعديلها وتطويرها.

نتبين من كل ما تقدم بأن الخوارزميات المتوازية هي فرع جديد وهام من فروع المعلوماتية ، كما أنها تفتح الأبواب على مصراعيها أمام الباحثين المهتمين في هذا المجال وإنني أدعو كافة الباحثين والمهتمين لمتابعة البحث سواء في المجال النظري أو في المجال العلمي لأنني أعتقد بأن هذا الفرع من فروع المعلوماتية سيبقى لوقت طويل موضوع الساعة ومحور اهتمام كافة المختصين والباحثين.

المصطلحات المستخدمة:

Account of cost	حساب الكلفة
Adaptive	متكيفة
Bit complexity	تعقيد الخانة
Broadcasting	الإعلان أو النشر
Chips	رقاقات
Circuit width	عرض الدارة
Concurrent	متعدد
Constant nodes	عقد ثابتة
Compilers	المصنّفات
Counting Steps	خطوات العد
Cube Connection	الاتصالات التكعيبية
Depth	عمق الدارة
Deterministic	حتمي
Efficiency	الفعالية
Exclusive	محدود أو مقصور
Feasibility	الوثوقية
Imperative	قاعدية
Input nodes	عقد الدخل
Interpretation	التفسير (الترجمة)
Linear Arraay	مصفوفة خطية
Lower and upper bounds	الحدود الدنيا والعليا
Multiple Instruction stream, Multiple Data stream (MIMD)	حواسيب MIMD (سلسلة تعليمات متعددة – سلسلة بيانات متعددة)
Multiple Instruction stream, Single Data stream (MISD)	حواسيب MISD (سلسلة تعليمات متعددة – سلسلة بيانات مفردة)
Multitasking	تعددية المهام
Operation nodes	عقد عمليات
Output nodes	عقد الخرج
Parallel time	زمن التوازي

Partial ordering graph	بيان الترتيب الجزئي
Perfect shffle connection	الاتصالات المختلطة التامة
Period of the circuit	مهلة الدارة
Prefix sums	المجاميع البادئة
Processor area	مجال المعالج
Quality	نوعية
Shared memory	ذاكرة مشتركة
Single Instruction stream, Multiple Data stream (MIMD)	حواسيب SIMD (سلسلة تعليمات مفردة – سلسلة بيانات متعددة)
Single Instruction stream, Single Data stream(SISD)	حواسيب SISD (سلسلة تعليمات مفردة، سلسلة بيانات مفردة)
Speed up	زيادة السرعة
String processors	معالجات خطية
R-block shared memory	ذاكرة مشتركة مقسمة إلى مقاطع
Recursive doubling	المضاعفة التعاودية
Reducing the number of processor	تخفيض عدد المعالجات
Routing steps	خطوات الاضطراب أو الدوران
Running time	زمن التنفيذ
Tow-Dimensional Array	مصفوفة ثنائية البعد
Tree Conection	شجرة الاتصال
Vector iterations	تكرارات المتجهات
Vectorization	طريقة استخدام المتجه
Vector machines	الآلات الشعاعية
Very large scale integration (VLSI)	مقياس الضم الكبير جداً
Wire length	طول السلك

قائمة الأشكال والجداول:

المراجع العلمية

1- [JeHk] – Jeffery H Kingston "Algorithms and Data structures" – Addison Wesley – 1997

2- [AnJe] - A.N.Aho , J.E.Hopcroft and j.D.Ulman "The Design And Analaysis Of computer Algorithms" - Addison Wesley - 1974

3- [SeGA] - Selim G.Akl " The Design & Analysis Of Parallel Algorithms " – prentice-Hall international, inc – 1989

4- [GSAG] – G.S.Almasi , A.Gottlieb "Highly Parallel Computing"- The Benjamin Cummings Publishing Company,Inc. – 1994

5- [MLFi] – M.L.Fisher "The Lagangian RelaxationMethod Of Solving The Multiconstraint 0-1 Knapsack To Optimality" – Managemment Science 27 -1981

6- [WShi] – W. Shih "A Branch And Bound Method For The Multi Constraint 0-1 Knasack Problem" - Journal of The Operational Research Society, 30(4) – 1979

7- [DFGP] – D. fayrd; G. Plateau " An Algorithm For The Solution Of The 0-1 Knapsak Problem" – Computing 28 - 1982

8- [TLCP] – T.L.Freeman , C.Philips " Parallel Numerical Algorithm" - Prentice Hall – 1993

9- [JHro] – J.Hromkovi "Communcation Complexity And Parallel Computing ; The Application Of Communication Comlexity In Parallel Computing" – Springerverleg , Berlin and Heidelberg GmbH & Co.KG,Feb.1997

10- [JHKi] – J.H.Kingston " Algorithms And Data Structures , Design, correctness , Analysis" – Addison-Wesley - 1990

11- [WaCD] – Ward Cheney,David Kincaid " NUMERICAL MATHEMATICS AND COMPUTING " – Brouks/cole publishing company.

12- [BrCM] – Brouno Codenotti,Mauro Leoncini "Introduction to Parallel Processing" – Addison Wesley Publishing company - 1992

13- [GC.O] - Gary Chartrand & Ortrud R.Oellermann – "APPLIED AND ALGORITHMIC GRAPH THEORY" – McGraw-Hill,Inc – 1993

14- [KMJm] – K.M.Chndy, J.Misra "Distributed Computation On Graphs :Shortest Path Algorithm" – CaCm , vol 25, n^0 11,Nov.1982

15- [IFos] - I.Foster " Designing And Building Parallel Programs " – Addison-Wesley – 1994

16- [JJAJ] – J.Ja'Ja " An Introduction To Parallel Algorithms " Addison-Wesley – 1992

17- [MJQu] - M.J.Quinn " Parallel Computing , Theory and Practice" McGraw-Hill , Series in Computer Science , Second edition,1994

18- [HELR] – H.El-Rewini " Distributed And Parallel Computing" – Prentice Hall ,Manning Publication, Series : Lecture Notes in Computer Science, vol.1220 -1998

19- [CSJJ] – C. Savage ; J.Ja'Ja " Fast Efficient Parallel Algorithms For some Graph problems" – SIAM J. On Computing, vol.16 n^0 4,Nov.1981

20- [KBag] – K. Bagchi "Advanced Computer Performance Modeling And Simulation" – Gordon and Breach, Published in UK, Oct. 1997

21- [ASSZ] – A. Stavrors; S.Zenios " Parallel Optimization, Theory, Algorithms And Applications" – Oxford University Press inc , Jan.1998

22- [RaRz] – الدكتور راكـــان رزوق "الخوارزميـــات وبنىالمعطيـــات" – منشورات المعهد العالي للعلوم التطبيقية والتكنولوجيا ، قسم المعلوماتيـــة - 1993

23- [MB.H] - الدكتورة ميســـاء بـــدر حمشـــو " الخوارزميـــات المتوازيـــة والخوارزميات الموزعة وتطبيقاتها في بحوث العمليات" – 1998

ملخص عن دراسة أعدت لنيل درجة الماجستير في المعلوماتية

بعنوان: (خوارزميات المعالجة المتوازية وبرمجتها)

إعداد الطالبة كندة زين العابدين

بإشراف الأستاذ الدكتور خالد علي الخنيفس

ومشاركة الدكتور حامد خالد الرجوب

تعريف تعددية المهام:

هي قدرة نظام التشغيل على تنفيذ أكثر من مهمة واحدة في الوقت نفسه ، وتبنى الخوارزمية في الأجهزة المتعددة المعالجات على أساس تعددية المهام والتنفيذ المتزامن وتعتمد البرمجة المتعددة المهام على فهم طريقة تنفيذ التعليمات على البيانات حيث إن هناك سلسلة من التعليمات (الخوارزمية) تأمر الحاسب بوظيفته في كل خطوة وهناك سلسلة من البيانات (دخل الخوارزمية) ستتأثر بتلك التعليمات.

[SeGA-1989] نستطيع أن نميز بين أربعة أنواع من الحواسيب:

1- أولاً: حواسيب SISD (سلسلة تعليمات مفردة، سلسلة بيانات مفردة)

Single Instruction stream, Single Data stream (SISD)

2- ثانياً: حواسيب MISD (سلسلة تعليمات متعددة – سلسلة بيانات مفردة)

Multiple Instruction stream, Single Data stream (MISD)

3- ثالثاً: حواسيب SIMD (سلسلة تعليمات مفردة – سلسلة بيانات متعددة)

Single Instruction stream, Multiple Data stream (SIMD)

4- حواسيب MIMD (سلسلة تعليمات متعددة – سلسلة بيانات متعددة)

Multiple Instruction stream, Multiple Data stream (MIMD)

في معظم المسائل الهامة التي نحلها باستخدام حواسيب من النوع SIMD من المستحب أن تكون المعالجات قادرة على الاتصال مع بعضها خلال الحساب من أجل تغيير البيانات أو النتائج المكررة. ونحقق هذا الاتصال وفقاً لفئتين فرعيتين: إحداهما يتم الاتصال فيها عن طريق الذاكرة المشتركة والأخرى عن طريق شبكة الاتصالات.

حواسيب SIMD (SM) بذاكرة مشتركة Shared-Memory :

[SeGA-1989] تعرف هذه الفئة أيضاً بآلـــة الوصـــول العشـــوائي المتـــوازي نمـــوذج (PRAM) (the Parallel Random-Access Machine)

عندما يريد معالجان الاتصال فإنهما يعملان خلال الذاكرة المشتركة ، ويمكن أن نقسم فئة الحواسيب هـــذه إلى أربع فئات فرعية وفقاً لقدرة اثنين أو أكثر من المعالجات على الوصول إلى موقع الذاكرة نفسه وبنفس الوقت وذلك كما يلي:

1- Exclusive-Read, Exclusive-Write (EREW) SM SIMD Computers.
2- Concurrent-Read, Exclusive-Write (CREW) SM SIMD Computers.
3- Exclusive-Read, Concurrent-Write (ERCW) SM SIMD Computers.
4- Concurrent-Read, Concurrent-Write (CRCW) SM SIMD Computers.

حواسيب SIMD شبكة اتصالات:

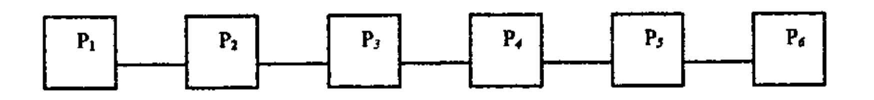
[SeGA-1989] يمكننا الحصول على نموذج مرن أكثر قوة من نموذج الذاكرة المشتركة يكون فيـــه كل زوج من المعالجات متصل بخط ثنائي الاتجاه ، حيث يمكن لعدة أزوج أن تتصل بآن واحد (ولكن بشرط ألا يحاول أكثر من معالج إرسال البيانات إلى معالج آخر أو تلقي البيانات من معالج آخر).

الشبكات البسيطة لحواسيب SIMD:

سنلخص بإيجاز الشبكات الأكثر شيوعاً منها فيما يلي:

1- مصفوفة خطية Linear Array:

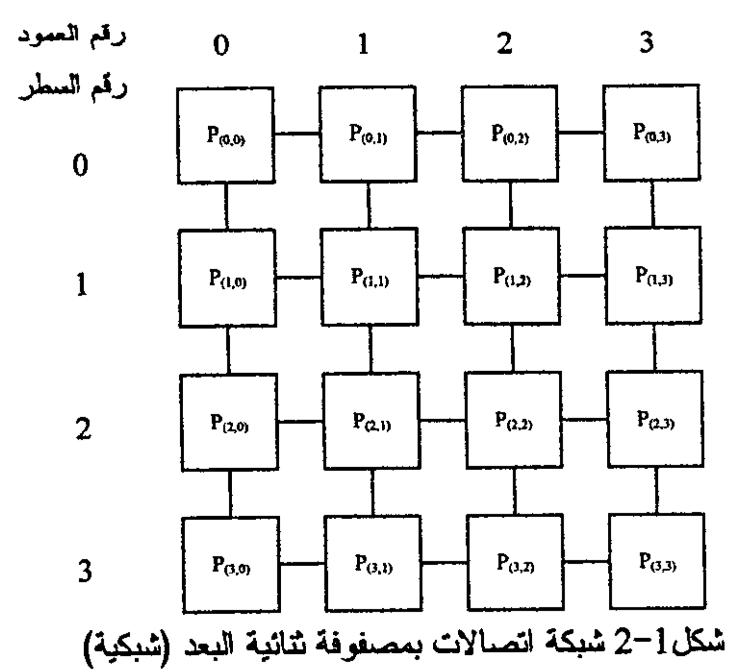
يمكن تمثيل وصل N معالج بشكل مصفوفة وحيدة البعد كما يظهر في الشكل (1-1) من أجل N=6. هنا المعالج P_i مرتبط بمجاوريه P_{i-1} و P_{i+1} من خلال خط اتصال ثنائي الاتجاه. كل من معالجـــات النهاية والمسماة P_1 و P_N تملك فقط مجاور واحد.



شكل (1-1) شبكة خطية من أجل N=6

2- مصفوفة ثنائية البعد Tow-Dimensional Array:

نحصل على شبكة ثنائية البعد بترتيب الـــ N معالج في مصفوفة m×m حيث $m = N^{1/2}$ كما يظهـــر الشكل(1-2) من أجل m=4

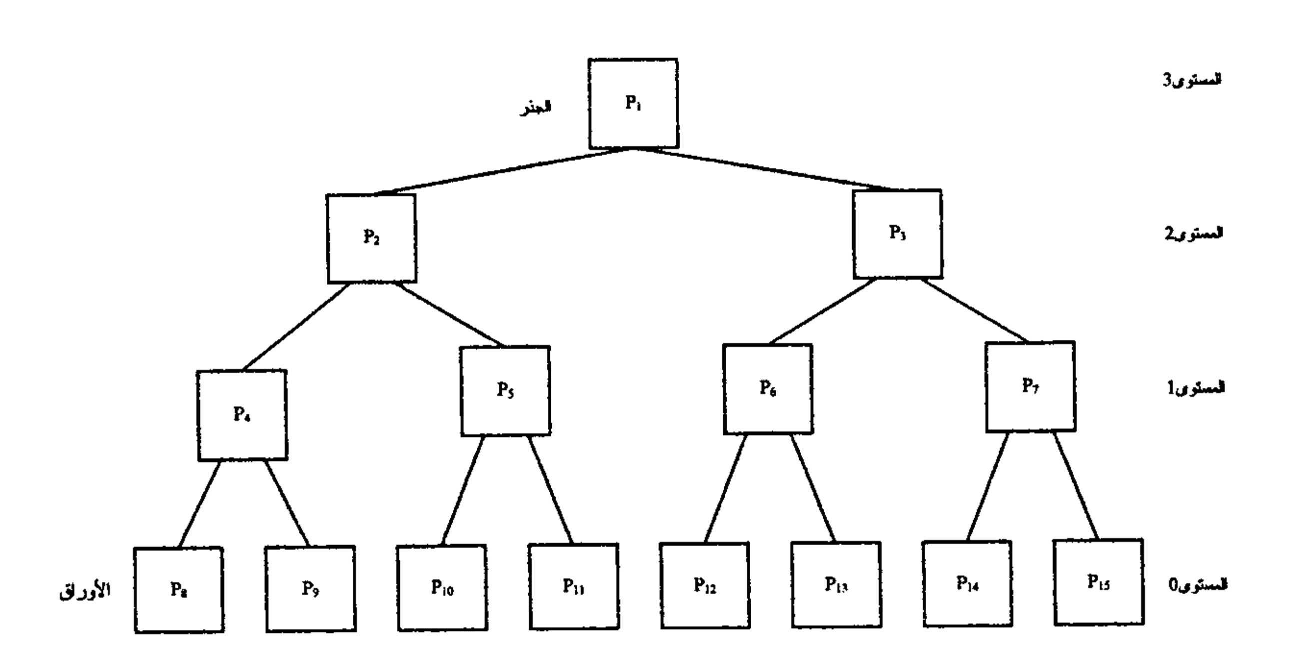


شكل1-2 شبكة اتصالات بمصفوفة ثنائية البعد (شبكية)

من أجل m=4

3- شجرة الاتصال Tree Connection:

تكون المعالجات في هذه الشبكة بشكل شجرة ثنائية تامة ومثل هذه الشجرة تملك d مستوى كما يظهر الشكل (3-1) من أجل d=4.



شكل (3-1) شبكة اتصالات شجرية من أجل d=4

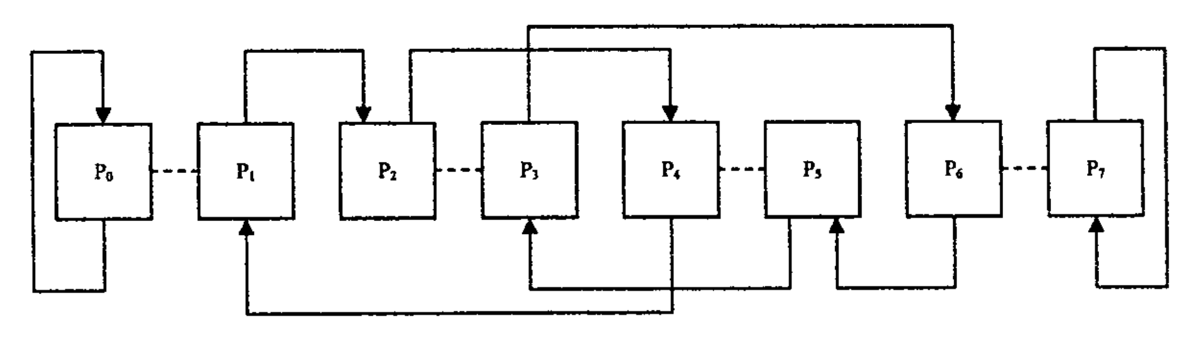
4- الاتصالات المختلطة التامة Perfect shuffle connection:

ليكن لدينا N معالج $P_0,P_1,P_2,..,P_{N-1}$ متوفرة حيث N هي قوة لـــ 2 :($N=2^x$)

يوجد في الاتصالات المختلطة التامة خط وحيد الاتجاه يربط P_i بـــ P_j حيث:

$$J=\begin{cases} 2i & \text{من أجل} \quad 0 <= I <= N/2 - 1 , \\ 2i+1-N & \text{من أجل} \quad N/2 <= I <= N-1 , \end{cases}$$

كما يُظهر الشكل(1-4) من أجل N=8

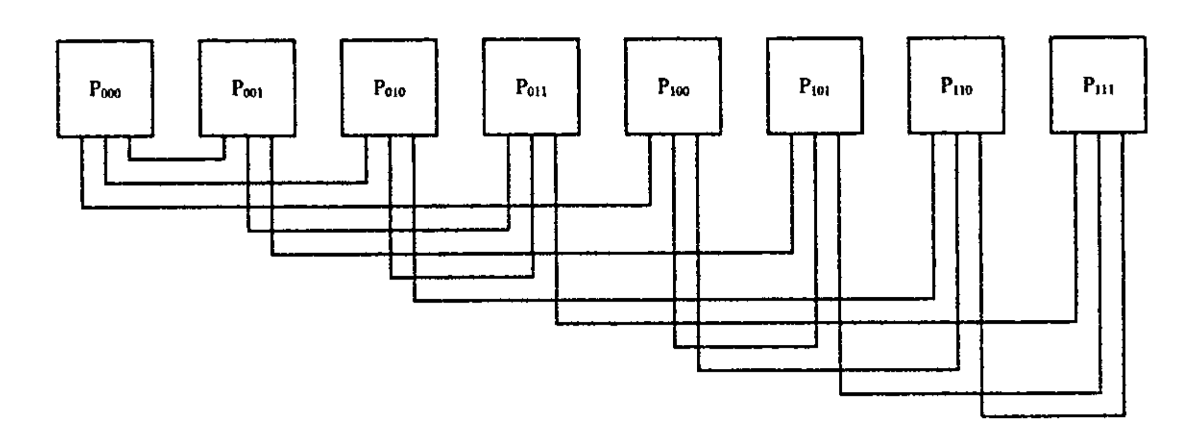


شكل (1-4) شبكة اتصالات مختلطة تامة من أجل N=8

5- الاتصالات التكعيبية Cube Connection:

لنفترض أن $N=2^q$ من أجل بعض q>=1 ولتكن الـــ N معالج: $P_0,P_1,P_2,..,P_{N-1}$ مكعب من q بعد أو (hypercube) نحصل عليه بوصل كل معالج إلى q مجاور.

ونرى شكل الشبكة فيما يلي من أجل q=3 والفهرس لـــ$p_0,p_1,..,p_7$ موضحاً بشكل رموز ثنائية.



شكل 1-5 شبكة اتصالات تكعيبية من أجل N=8

نظريات الحساب المتوازي:

[MJQu,1994] تعرّف نظريات الحساب المتوازي أن المكان والزمان هما المصـــدران الأساســيان الأكثر أهمية المعرفان لتعقيد الحساب التسلسلي. وفي الحساب المتوازي تبقى أهمية الزمن نفسها: حيث أن السبب الرئيسي لدراسة الحساب المتوازي هو للحصول على زيادة سرعة حل المســائل الحســابية speed up ، وإذا توفر لدينا وحدات معالجة أكثر فإن الزمن المطلوب لحل مسألة معطـــاة يـــنخفض، وهذه الحقيقة تجعل وحدات المعالجة هي المتطلب الرئيسي الثاني في الحساب المتوازي.

الحسابات المتوازية والنماذج اللوغاريتمية:

تعريف النموذج اللوغاريتمي:

[MJQu,1994] يمكن وصفه بالحقائق التالية:

5- يمكن أن يستعمل أي عدد من المعالجات في أي لحظة.

6- يمكن أن ينفذ كل معالج أي تعليمة (رياضية أو منطقية) في وحدة زمن.

7- لا يوجد كلفة من أجل الوصول للبيانات.

8- لا يوجد كلفة للاتصال بين المعالجات.

أما المصادر الأساسية المطلوبة من النموذج فهي عدد الخطوات المتوازية (عدد وحدات الزمن) وعدد المعالجات المستعملة كدالة في حجم المسألة.

الدارات المنطقية:

[JJAJ,1992] , [MJQu,1994] إن الحواسيب الاعتيادية هي آلات متوازية كما أن كـــل حاســب مصنوع من دارات الكترونية تحوي الملايين من العناصر المفتاحية وخطوط النقل كلها تعمـــل علـــى التوازي ، والدارات المنطقية هي نموذج منطقي من الدارات الالكترونية. يعمل هذا النمـــوذج علـــى مسائل تمثل بسلسلة من الخانات ، تحول جميع المعلومات الداخلة والخارجة من حاسب حقيقـــي إلـــى رموز في سلسلة من الخانات الثنائية، ومع بعض الترتيب في البنية تصبح دالة على بيانات ، بمعنى أن السلسلة تعتمد على التفسير (الترجمة) interpretation. وبما أن تنفيذ العمليات يتم في مستوى الخانة فإنه عندما نعبر عن خوارزمية بدارة منطقية فإن كلفتها تدعى [كلفة الخانة bit cost] بينمـــا يـــدعى تعقيدها [تعقيد الخانة bit complexity] وهي تعد مقياساً لكلفة التوازي. إن الخصائص السابقة تجعل نموذج الدارة المنطقية قاعدة لدراسة تعقيد الحسابات المتوازية.

تعريف الخوارزمية المتوازية:

تعرف الخوارزمية المتوازية بأنها مجموعة من العمليات التي تنجز كل منها خوارزمية تتابعية خاصة بها وتتبادل الاتصالات فيما بينها عبر شبكة اتصالات محددة.
وهي تتبع بشكل أساسي للنماذج الرياضية الأساسية في الحاسوب المتوازي والتي قمنا بتعريفها في الفصل الأول وتتمايز أنواعها حسب أنواع الحواسيب المتوازية.

تحليل الخوارزميات المتوازية:

[SeGA-1989] , [AnJe-1974] , [GSAG-1994] يحدد تحليل الخوارزميات المتوازية درجة جودة الخوارزمية وهذا يعني سرعتها وكلفة تنفيذها ومدى فعاليتها عند استعمالها في الوسائل المتاحة.
ولذلك فإننا نهتم بالمعايير التالية: زمن التنفيذ وعدد المعالجات المستخدمة ومن ثم نقوم بحساب الكلفة.

أولاً: زمن التنفيذ Running time

هو الزمن الذي تأخذه الخوارزمية خلال حل المسألة على حاسب متوازي. وذلك يعني أنه الزمن المستهلك من الخوارزمية منذ اللحظة التي تبدأ فيها إلى اللحظة التي تنتهي فيها.

ثانياً: عدد المعالجات

المعيار الثاني الأهم في تقييم الخوارزمية المتوازية هو عدد المعالجات المطلوبة في حل مسألة. حيث أنها مكلفة من أجل الشراء والصيانة وتشغيل الحواسيب.

ثالثاً: حساب الكلفة ACCOUNT OF COST

تعرف كلفة خوارزمية التوازي على أنها حاصل ضرب المعيارين السابقين:

الكلفة = زمن التنفيذ × عدد المعالجات المستخدمة

إذاً الكلفة تساوي عدد الخطوات المنفذة بشكل جماعي من قبل المعالجات في حل مسألة بأسوأ حالة. يفترض التعريف أن جميع المعالجات تنفذ نفس عدد الخطوات. وإذا لم تكن الحالة كذلك فإن الكلفة هي الحد الأعلى في ناتج عدد الخطوات المنفذة.

تصميم الخوارزميات المتوازية:

[IFos-1994] , [JeHk-1997] , [BrCMal-1992] , [JHKi-1990] نبدأ بتطوير الخوارزميات المتوازية من الخوارزمية التسلسلية ومن ثم نحدد الخطوات المستقلة وأخيراً نحصل على الخوارزمية المتوازية. وهذا التحول من التسلسل إلى التوازي ينتج غالباً خوارزميات متوازية ذات كلفة منخفضة.

ومن أجل ذلك تنشأ مسألة تحديد استراتيجيات للحصول على خوارزميات متوازية فعالة حيث لا يوجد طريقة عامة تعطي أفضل خوارزمية متوازية لمسألة معطاة ولكن هناك تقانات تستعمل لهذا الغـرض وسنعرض بعض هذه الطرائق والتقانات:

6. التقسيم والتخصيص Divide et impera:

تعتمد هذه التقانة على تقسيم المسألة إلى مسائل جزئية من نفس النوع ولكن بحجم أصغر ومن ثـم إعادة ترتيب وتجميع الحلول. وهي طريقة فعالة جداً حيث تسمح بتجزئة المسألة إلى مسائل جزئية مستقلة تُحل على التوازي وتُجمع حلولها للحصول على حل المسألة الأصلية.

7. طريقة استخدام المتجه Vectorization:

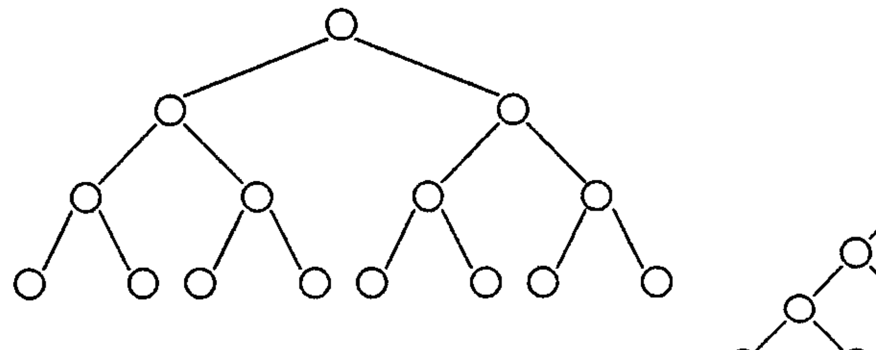
تبدأ هذه التقانة من تحليل المسألة وإنتاج الخوارزميات التي تعمل على بنى المعطيات التي تتناسب مع حسابات المتجه (المتجهات والمصفوفات).

8. تكرارات المتجهات vector iterations:

قام بوضع هذه التقانة Traub وآخرون من أجل حل الجمل الخطية المثلثية. حيث تبـدأ باسـتخدام الخوارزميات التسلسلية المباشرة وتبنى منها الطريقة التكرارية المتوازية وهكذا..

9. المضاعفة التعاودية Recursive doubling:

وهي تتألف من تحويل البيان الحسابي من مثل البيان المعروض في الشكل (2-1 -أ-) إلى بيـان من الشكل(2-1 -ب-) وهذا يمكننا من الحصول على بيان تكون فيه كلفة التجوال (درجة التعقيد) تابع أسي (لوغاريتمي) من بيان فيه كلفة التجوال (درجة التعقيد) تابع خطي.



الشكل(2-1) -أ- بيان بدرجة تعقيد خطية

الشكل(2-1) -ب- بيان بدرجة تعقيد لوغاريتمية

10. تخفيض عدد المعالجات Reducing the number of processor:

حيث يمكن في بعض الأحيان إنقاص عدد المعالجات الكبير باستعمال خوارزمية متوازية وذلك بزيادة زمن التنفيذ بواسطة عامل ثابت.

كفاءة خوارزمية:

[JHro-1997] , [TLCP-1993] , [WaCDaK] نعرف كفاءة أو كلفة خوارزمية بأنها ناتج ضرب عدد المعالجات التي تستعمل لإنجاز العملية وزمن التنفيذ للعملية. إن كلفة حل الجمل الكبيرة من المعادلات الخطية على الحاسب كبيرة ولمعرفة كيفية حساب الكلفة لخوارزمية ما لها كود معطى نقوم بعد عمليات الضرب والقسمة (العمليات الطويلة) لأن لها أكبر زمن مستهلك، وبعد ذلك نجمّع عمليات الضرب والقسمة معاً حتى ولو كانت عمليات القسمة أبطأ من عمليات الضرب.

مقارنة بين الخوارزميات التسلسلية والمتوازية:

[JeHk-1997] , [RaRz-1993] يفيد تحليل الخوارزميات في مقارنة خوارزميات حل مسألة معينة فما نريد الوصول إليه هو ما يلي: مهما تكن الآلة التي ستنفذ الخوارزمية ومهما تكن لغة البرمجة المستخدمة فإن: لنفرض أن لدينا خوارزميتين A و B تنفذان المهمة البرمجية نفسها فإن الخوارزمية A أفضل من الخوارزمية B.

خوارزمية التحقق من الانتهاء من التوزيع (خوارزمية النشر):

قد نحتاج أحياناً إلى نشر عنصر بيانات ما على جميع المعالجات في الحاسب المتوازي من أجل استعماله من قبلها جميعاً في الوقت نفسه خلال تنفيذ الخوارزمية. وهذه حالة خاصة من عملية القراءة المتعددة العامة. ومن الواضح أن المحاكاة الفعالة لهذه العملية لا يمكن إنجازها في خطوة واحدة على نموذج EREW ونحتاج من أجل تلك المحاكاة إلى وضع إجراء ينفذ هذه المهمة. وقد قمنا بشرح هذه العملية بالإجراء BROADCAST.

خوارزمية مراقبة الانتهاء:

قد نحتاج أحياناً إلى معرفة ما إذا أنهت هذه المعالجات مهمتها أم لا ويتم ذلك كما يلي:

لنأخذ موقع f ، يحتوي هذا الموقع على قيمة منطقية تحدد فيما إذا أتم أحد المعالجات المهمة المطلوبة وبالتالي تنهي باقي المعالجات عملها. في بداية الأمر تأخذ f القيمة false وعندما ينهي المعالج مهمته

فإنه يضع في f القيمة true ، وفي كل خطوة تقوم جميع المعالجات بفحص قيمة f فإذا كانت true تتوقف عن العمل.

في حالة الحواسيب من النموذج EREW نلاحظ أن هذه العملية تحتاج إلى log N خطوة لنشر قيمة f في كل مرة تحتاج إليها المعالجات وهذا يزيد زمن التنفيذ. أما في نموذج الحواسيب CREW فــإن عمليات القراءة المتعددة مسموحة وبالتالي تأخذ خطوة واحدة لجميع المعالجات من أجل الحصول على قيمة f مما يخفض من زمن التنفيذ. كما نلاحظ أنه ربما ينهي أحياناً أكثر من معالج مهمتــه فــي آن واحد فيحتاج عندئذ أكثر من معالج إبلاغ النجاح في الوقت نفسه وهذا يعني أن اثنــين أو أكثــر مــن المعالجات ستحاول الكتابة في الموقع f في الوقت ذاته وهذه العملية ممكنة فــي نمــوذج الحواسـيب CRCW SM SIMD فقط.

خوارزمية سبر التفرع الشجري:

[GC.O,1993], [KMJm,1982] , [CSJJ,1981] يعتبر البحث أحد العمليات الشائعة في مجال الحسابات. ويستخدم في التطبيقات التي نحتاج فيها لمعرفة فيما إذا كان عنصر ما ينتمي إلى قائمة أو بشكل أعم تكرار في معلومات حقل مرتبطة بذلك العنصر.

تتوضح مسألة البحث بشكل قاعدة عامة كما يلي:

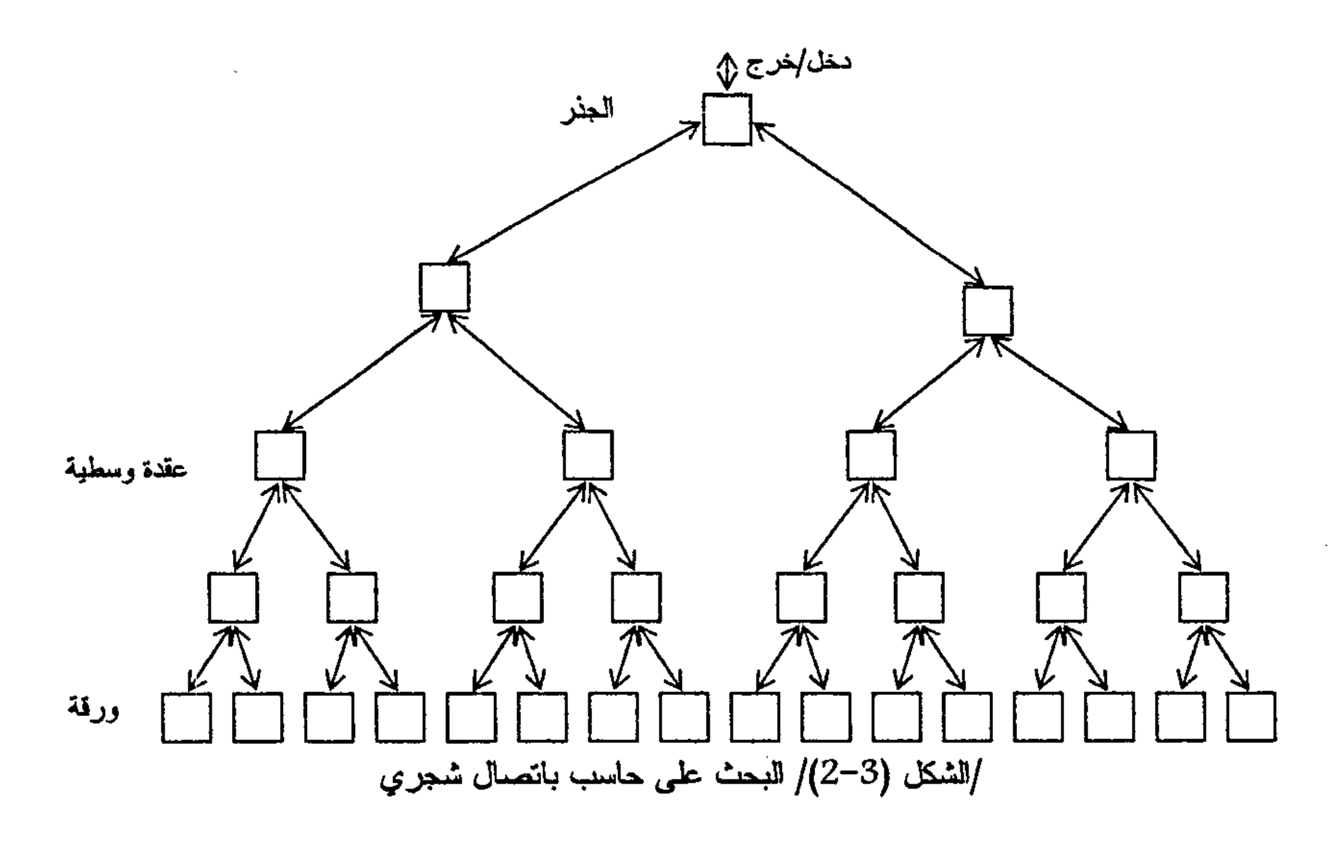
تعطى متسلسلة $S=\{s_1,s_2,.........s_n\}$ من الأعداد الصحيحة وعدد صحيح x والمطلوب تحديد فيما إذا كان $s_k = x$ حيث s_k عنصر من S. تحل هذه المسألة في الحسابات التسلسلية بمسح المتسلسلة S ومقارنة العنصر x مع عناصرها المحددة وتستمر هذه العملية حتى نجد عنصراً صحيحاً مساوياً لــx أو تعود العملية بالفشل.

لنفترض الآن أن لدينا حاسب SIMD باتصال شجري مع n مستوى من أجل البحث في ملف فيه n سجل. مثل الشجرة المبينة في الشكل من أجل $n=16$.

تختلف مهام المعالجات في هذا النموذج حسب توضعها في الشجرة :

- تخزن كل ورقة في الشجرة سجل واحد للملف الذي نبحث فيه.
- يتلقى الجذر الإدخال من العالم الخارجي ومن ثم يمرر ناتج الخرج الذي يتلقاه من أبنائه إلى العالم الخارجي، وبالعكس يتلقى الخرج من أبنائه ويصدره إلى العالم الخارجي.
- أما من أجل العقد الوسطى فكل منها يستطيع القيام بما يلي:

1. تتلقى العقدة الوسطى إدخالاً واحداً من الخلف (الأب) ويصنع منه نسختين يرسل واحدة إلى كل من أبنائه.
2. يتلقى إدخالين من ابنيه يقارنهما ويرسل النتيجة إلى والده.



/الشكل (3-2)/ البحث على حاسب باتصال شجري

تطبيقات الخوارزميات المتوازية:

سنعرض الآن بعض تطبيقات مسائل القرار والأمثلية التي نستخدم فيها خوارزميات متوازية فعالة مثل خوارزمية حساب المجاميع البادئة لسلسلة وقد اخترنا مسألتين هما:

مسألة الأعمال التسلسلية وفق فترات انتهائها (job sequencing with dead lines) وهي مسألة قرار و مسألة حقيبة الظهر (knapsack problem) وهي مسألة حل أمثل.

لدينا في كل من المسألتين خوارزمية تعمل على حاسب متوازي باتصال شجري ، علماً أن الخطوة الأساسية في كل من الخوارزميتين هي حساب المجاميع البادئة لسلسلة، وقد تم عرض عدة خوارزميات لمسألة حساب المجاميع البادئة من أجل عدة أنواع من الحواسيب.

1- خوارزمية الإجراء : SEQUENTIAL SUMS(X,S) من أجل الحالة التسلسلية.

2- خوارزمية الإجراء : PARALLEL SUMS(X,S) من أجل حاسب متعدد المعالجات.

3- خوارزمية الإجراء ALLSUMS(a_1,a_2,..,a_N) من أجل حساب جميع المجاميع.

4- خوارزمية المجاميع البادئة على شجرة.

5- المجاميع البادئة على الشبكة Mesh.

مسألة الأعمال المتسلسلة وفق فترات انتهائها :

ليكن لدينا مجموعة من الأعمال نريد تنفيذها على آلة واحدة وليكن عددها n ولنرمز لها بالشكل التالي: $J = \{j_0, j_1, \ldots, j_{n-1}\}$. يمكن للآلة أن تنفذ عملاً واحداً في زمن ما وعندما تحدد عملاً فيجب عليها أن تكمله قبل أن تبدأ بمعالجة العمل التالي.

يرتبط بكل عمل j_i ما يلي:

3- زمن معالجة t_i .

4- مدة انتهاء d_i يجب أن يكتمل العمل من خلالها.

نعرف جدول الأعمال بأنه تبديل الأعمال في J الذي يحدد ترتيب تنفيذها، ويقال عن الجدول أنه مناسب أو عملي إذا كان كل عمل ينتهي بمدة انتهائه .

والمطلوب أنه إذا كان لدينا n عمل $\{j_0, j_1, \ldots, j_{n-1}\}$ لها أزمنة المعالجة $\{t_0, t_1, \ldots, t_{n-1}\}$ ولها فترات الانتهاء $\{d_0, d_1, \ldots, d_{n-1}\}$ فهل يوجد جدول عملي لهذه الأعمال .

يمكن الإجابة عن هذا السؤال بالإيجاب إذا وفقط إذا كان أي جدول أعمال تنفذ فيه الأعمال بترتيب متزايد لفترات الانتهاء عملياً. لذلك يكفي لحل المسألة إيجاد جدول أعمال ترتب فيه الأعمال بترتيب متزايد لفترات الانتهاء وفحص فيما إذا كان هذا الجدول عملياً أم لا وفي حال تم ذلك فإن الإجابة عن السؤال تكون نعم و إلا فالإجابة لا.

تحتاج هذه الخوارزمية لكلفة زمنية $O(n \log n)$ لترتيب الأعمال بشكل تسلسلي ومن ثم إلى كلفة زمنية $O(n)$ لاختبار فيما إذا كان كل عمل يمكن أن يتم في فترة انتهائه.

مسألة حقيبة الظهر THE KNAPSACK PROBLEM:

[DFGP,1982] , [WShi,1979] ,[MLFi,1981] في هذه المسألة تتميز حقيبة الظهر بأنها تستطيع حمل وزن أعظمي W ومجموعة من n مادة ولتكن: $A=\{a_0, a_1, \ldots, a_{n-1}\}$ والتي أوزانها على التوالي: $\{w_0, w_1, \ldots, w_{n-1}\}$ كما يرتبط بكل مادة فائدة ونرمز لمجموعة الفوائد بــ: $\{p_0, p_1, \ldots, p_{n-1}\}$، إذا وضعنا في حقيبة الظهر الجزء z_i من المادة التي وزنها w_i حيث $0 \le z_i \le 1$ عندئذ فإن الفائدة المكتسبة هي $z_i p_i$. والمطلوب في هذه المسألة هو تعبئة حقيبة الظهر بالمواد أو بأجزاء منها بحيث:

3- الوزن الناتج للمواد المحددة لا يتجاوز W

4- الفائدة المكتسبة الناتجة أكبر ما يمكن.

درسنا حلول هذه المسألة على حاسب متوازي متصل شجرياً ومن ثم على حاسب متوازي متصل شبكياً.

خوارزميات متوازية لأجهزة متعددة المعالجات ذات ذاكرة مشتركة:

سنقوم فيما يلي بعرض بعض المسائل الشهيرة رياضياً ، ونضمن التعاريف الخاصة بها، ونضع خوارزميات الحلول التسلسلية والمتوازية لها ، ونميز بين الخوارزميات المتوازية من أجل النماذج المختلفة للحواسيب وطرائق اتصالها في بعض منها ، ومن ثم نعرض تحليلاً لكل منها لدراسة أمثليتها وكلفتها الزمنية وغيرها مع ذكر بعض الأمثلة التوضيحية.

سنأخذ مسألتين من مسائل المقارنة مسألة الاختيار في سلسلة التي ظهرت في العديد من تطبيقات علوم الحاسبات والإحصاء، ومن ثم نأخذ مسألة أخرى تعرف بمسألة الدمج (لدمج متسلسلتين من البيانات لتشكيل متسلسلة ثالثة جديدة) وهي أحد مسائل المقارنة فقد ازداد الدمج في علم الحاسبات مع تنوع المحتويات متضمناً قواعد البيانات بشكل خاص وترتيب الملفات عموماً إذ أن العديد من هذه التطبيقات تستعمل مسألة الدمج للبيانات الرقمية ومن الضروري توظيف الدمج بشكل تام بحيث يحذف التكرار في الإدخالات من المتسلسلة الناتجة.

مسألة الاختيار:

تحتوي دراستنا عن تحليل وتصميم الخوارزميات على المسألة التالية:
لنأخذ سلسلة S من n عنصر وعدد صحيح k حيث 1<k<n والمطلوب تحديد العنصر k الأصغر في S //تعرف هذه المسألة بمسألة الاختيار//.
وغرضنا هنا هو إيجاد الخوارزمية المتوازية لحل هذه المسألة على حواسيب من النموذج SIMD بذاكرة مشتركة. وسنصمم الخوارزمية بحيث تلائم عدداً من الأهداف ومن ثم نحللها لنتحقق من أن تلك الأهداف ملائمة فعلاً.
تنتمي هذه المسألة إلى مجموعة مسائل تدعى مسائل المقارنة /comparison problems/ حيث تحل هذه المسائل عادة عن طريق مقارنة أزواج من العناصر في سلسلة مدخلة.
بدأنا في هذا المقطع بالتعريف النظامي لمسألة الاختيار وعرض الحد الأدنى لعدد الخطوات المطلوبة لحل المسألة على حاسب تسلسلي. وهذا يؤدي إلى الحد الأدنى لكلفة أي خوارزمية متوازية لمسألة الاختيار. عرضنا خوارزمية تسلسلية أمثلية وتتمثل أهداف تصميمنا على تحقيق الخواص المطلوبة بشكل عام في أية خوارزمية متوازية. ومن ثم قمنا بوصف خوارزمية الاختيار المتوازي وتحليلها.

مسألة الدمج:

وهي أحد مسائل المقارنة ويمكن أن نعرفها كما يلي: لتكن $A=\{a_1,a_2,..,a_s\}$, $B=\{b_1,b_2,..,b_r\}$ متسلسلتين من الأرقام المرتبة تصاعدياً والمطلوب دمج كلاً من المتسلسلتين A , B لتشكيل متسلسلة ثالثة C: $C = \{c_1,c_2,..,c_{r+s}\}$ مرتبة تصاعدياً. حيث أن كل من العناصر c_i في C تنتمي إما إلى A أو إلى B وكل من العناصر a_i و b_i تظهر مرة واحدة فقط في C.

شبكة الدمج:

يمكننا أن نحصل على شبكة غرض خاص لإنشاء التوازي بأي من الطرق التالية:

4- استعمال معالجات متخصصة مع شبكة الاتصالات التقليدية.

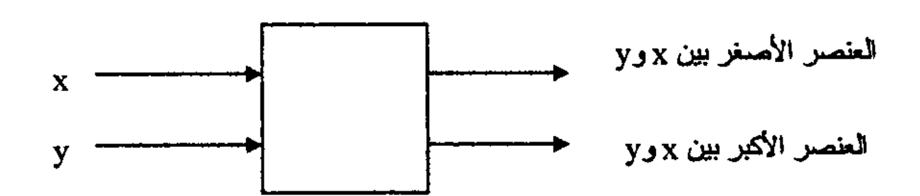
5- استعمال شبكة اتصالات مصممة بشكل عادي لربط معالجات قياسية.

6- استعمال خليط من 1و2.

سنأخذ في دراستنا لهذه المسألة الحالة الثالثة حيث سيتم الدمج عن طريق مجموعة من المعالجات البسيطة جداً متصلة بواسطة شبكة غرض خاص وهو البناء المتوازي وتعرف بـــ:

شبكة الدمج (r,s) /(r,s)-merging network/

إن جميع المعالجات المستعملة متماثلة وتدعى المقارنات comparators. يتلقى المقارن إدخالين ويصدر إخراجين كما يوضح الشكل (3-9).



الشكل (3-9) يوضح دخل وخرج المعالج المقارن

العملية الوحيدة التي يستطيع المقارن إنجازها هي مقارنة قيم الإدخالين الخاصين به ومن ثم وضع الأصغر منهما في قمة خطوط الخرج والأكبر منهما في أسفلها.

قمنا بداية بوصف بناء الشبكة الخاصة للدمج، ومن ثم أوجدنا خوارزمية متوازية تعمل على النموذج CREW SM SIMD وهي متكيفة وأمثلية، وبما أن الخوارزمية تستخدم الخوارزمية التسلسلية للدمج فقد قمنا بشرحها كإجراء. وهي توضح عمليات القراءة المتعددة التي يمكننا إزالتها من الخوارزمية المتوازية عن طريق المحاكاة على حاسب من النموذج EREW. وقد أوجدنا خوارزمية متكيفة وأمثلية

تعمل على النموذج EREW SM SIMD لها زمن تنفيذ أصغر من تلك المحاكاة وقد اعتمدت الخوارزمية على إجراء تسلسلي لإيجاد العنصر الأوسط في متسلسلتين مرتبتين.

النتائج:

- إن دراسة الخوارزميات المتوازية هذا الفرع الجديد من فروع المعلوماتية مسألة صعبة في وقتنا الراهن وخصوصاً لعدم توفر أجهزة ملائمة تسمح بترجمة الخوارزميات المقدمة في هذه الرسالة إلى برامج عملية نتحقق من خلال تطبيقها على مسائل واقعية من أهمية النتائج التي يمكن أن نتوصل إليها.
- بينّا في الفصل الأول من الرسالة مفهوم الأنظمة متعددة المعالجات وتعريفها وتصنيفها ودراسة أسلوب التواصل فيها ، كما قدمنا أفكاراً ونتائج تشكل أساسات نظرية الحساب المتوازي و وضحنا بعض نماذج الحساب المتوازي.
- درسنا في الفصل الثاني الخوارزميات المتوازية وأنواعها وميزنا خواصها وتقنيات تصميمها ومقاييس تحليلها وتحديد كفاءتها.
- قدمنا في الفصل الثالث مسائل هامة وخوارزميات حلها بشكل متوازي وذلك بالاعتماد على أجهزة متعددة المعالجات ، ووجدنا أننا نستطيع باستخدام الخوارزميات المتوازية زيادة حجم المسائل الممكن حلها كما تساعد الخوارزميات المتوازية في الحصول على معلومات أكثر وحلول أفضل بتقليل الكلفة للمسائل كما وجدنا ضرورة تخصيص المعالجات بحجم كبير من العمل لاستثمار الحواسيب المتوازية بشكل أفضل.
- تسمح المسائل التي تمت معالجتها في هذا الفصل في توضيح أثر الموازاة في زيادة حجم المسائل التي يمكن حلها وتقليل زمن الوصول إلى الحل الأمثل إذ أن استخدام خوارزمية التحقق من الانتهاء يخبر بوصول مبكر للحل وبالتالي يؤدي إلى تقليل زمن التنفيذ.

المقترحات والتوصيات:

- ❖ أقترح إجراء دراسة عامة وموسعة لكافة المسائل بالأسلوب المتوازي وفهم طرائق التعامل مع الأجهزة المتوازية وشبكات اتصالها لتحديد الخوارزميات المناسبة لحل المسائل.
- ❖ تبين لنا أن المسألة الملحة الواجب معالجتها من وجهة نظر البرمجيات تكمن في تخصيص المهمات أو العمليات للمعالجات وتطرح هذه المسألة نفسها في الأجهزة المتعددة المعالجات ولابد من إجراء دراسات متعمقة في هذا المجال حتى نحصل على أفضل النتائج المأمولة من الموازاة في البرمجة.

❖ أقترح الاستفادة من مفاهيم وخواص الخوارزميات المتوازية لوضع خوارزميات جديدة أو تطوير الخوارزميات القديمة لحل المسائل التي لا نعرف حلاً عاماً لها.

❖ أرى أن مشكلة نقل المعلومات التي ظهرت في مسألة البحث الشجري وتظهر في العديد من المسائل المشابهة ستبقى لمدة طويلة موضوع الساعة ومن المناسب التعمق في دراستها والبحث عن خوارزميات تسمح بمعالجتها وتعديلها وتطويرها.

المراجع العلمية:

- [SeGA] - Selim G.Akl " The Design & Analysis Of Parallel Algorithms " – prentice-Hall international, inc – 1989
- [MJQu] - M.J.Quinn " Parallel Computing , Theory and Practice" McGraw-Hill , Series in Computer Science , Second edition,1994
- [DFGP] – D. fayrd; G. Plateau " An Algorithm For The Solution Of The 0-1 Knapsak Problem" – Computing 28 - 1982
- [WShi] – W. Shih "A Branch And Bound Method For The Multi Constraint 0-1 Knasack Problem" - Journal of The Operational Research Society, 30(4) – 1979
- [MLFi] – M.L.Fisher "The Lagangian Relaxation Method Of Solving The Multiconstraint 0-1 Knapsack To Optimality" – Managemment Science 27 -1981
- [CSJJ] – C. Savage ; J.Ja'Ja " Fast Efficient Parallel Algorithms For some Graph problems" – SIAM J. On Computing, vol.16 n^0 4,Nov.1981
- [KMJm] – K.M.Chndy, J.Misra "Distributed Computation On Graphs :Shortest Path Algorithm" – CaCm , vol 25, n^0 11,Nov.1982
- [GC.O] - Gary Chartrand & Ortrud R.Oellermann – "APPLIED AND ALGORITHMIC GRAPH THEORY" – McGraw-Hill,Inc – 1993
- - 1993
- [WaCDaK]– Ward Cheney,David Kincaid " NUMERICAL MATHEMATICS AND COMPUTING " – Brouks/cole publishing company.
- [TLCP] – T.L.Freeman , C.Philips " Parallel Numerical Algorithm" - Prentice Hall – 1993
- [JHro] – J.Hromkovi "Communcation Complexity And Parallel Computing ; The Application Of Communication Comlexity In Parallel Computing" – Springerverleg , Berlin and Heidelberg GmbH & Co.KG,Feb.1997
- [IFos] - I.Foster " Designing And Building Parallel Programs " – Addison-Wesley – 1994
- [JeHk] – Jeffery H Kingston "Algorithms and Data structures" – Addison Wesley – 1997

- [BrCM] – Brouno Codenotti,Mauro Leoncini "Introduction to Parallel Processing" – Addison Wesley Publishing company - 1992
- [JHKi] – J.H.Kingston " Algorithms And Data Structures , Design, correctness , Analysis" – Addison-Wesley - 1990
- [AnJe] - A.N.Aho , J.E.Hopcroft and j.D.Ulman "The Design And Analaysis Of computer Algorithms" - Addison Wesley – 1974
- [RaRz] – الدكتور راكان رزوق "الخوارزميات وبنىالمعطيات" – منشورات المعهد العالي للعلوم التطبيقية والتكنولوجيا ، قسم المعلوماتية

Abstract

Multitasking:
Multitasking is the ability of the computation system to executing more than one task in the same time.
The algorithm in the multiprocessor computers is built on the multitasking and the simultaneous executing.
[SeGA-1989] Multitasking programming depends on the understanding of the method of executing instructions on data. A stream of instructions (the algorithm) tells the computer what to do at each step. A stream of data (the input of the algorithm) is affected by these instructions.
We can distinguish among four classes of computers:

1. SISD Computers: single instruction stream , single data stream
2. MISD Computers: Multiple instruction stream , single data stream
3. SIMD Computers: single instruction stream , Multiple data stream
4. MIMD Computers: Multiple instruction stream , multiple data stream

SIMD Computers:
In the most interesting problems that we wish to solve on an SIMD Computers , it is desirable for the processors to be able to communicate among themselves during the computation in order to exchange data or intermediate results. This can be achieved in two ways , giving rise to two subclasses : SIMD computers where communication is through a shared memory and those where it is done via an interconnection network.
Shared- Memory (SM) SIMD Computers:
[SeGA-1989] This class is also known in the literature as the Parallel Random – Access Machine (PRAM) model.
During the execution of a parallel algorithm the N processors gain access to the shared memory. However , the class of shared- memory SIMD computers can be further divided into four subclasses, according to whether two or more processors can gain access to the same memory location simultaneously:

1. Exclusive- Read , Exclusive- Write (EREW)SM SIMD Computers.
2. Concurrent- Read , Exclusive- Write (CREW)SM SIMD Computers.
3. Exclusive - Read , Concurrent - Write (ERCW)SM SIMD Computers.
4. Concurrent- Read , Concurrent - Write (CRCW)SM SIMD Computers.

Interconnection- Network SIMD Computers:
[SeGA-1989] This model is more powerful than the R-block shared memory , as it allows instantaneous communication between any pair of processors. Several pairs can thus communicate simultaneously (provided, of course, no more than

one processor attempts to send data to or expects to receive data from another processor).

Simple Networks For SIMD Computers:

The most popular of these networks are briefly outlined in what follows:

(i) Linear Array:

The simplest way to interconnect N processors is in the form of a one-dimensional array, as shown in Fig. 1.1 for N=6. Here, processor P_i is linked to its two neighbors p_{i-1} and p_{i+1} through a two-way communication line. Each of the end processors, namely, p_1 and p_N, has only one neighbor.

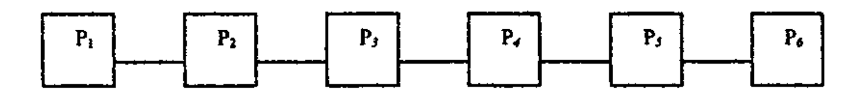


Figure 1.1 Linear array connection

(ii) Two-Dimensional Array:

A two-dimensional network is obtained by arranging the N processors into an m×m array, where $m=N^{1/2}$, as shown in Fig 1.2 for m=4.

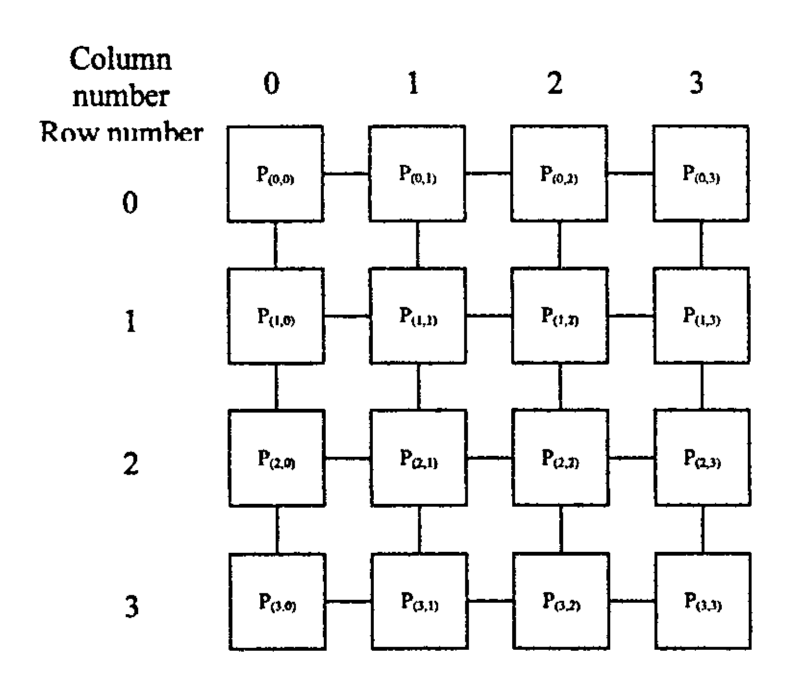


Figure 1.2 Two- dimensional array (or mesh) connection

(iii) Tree Connection:

In this network, the processors form a complete binary tree. Such a tree has d levels, as shown in Fig. 1.3 .

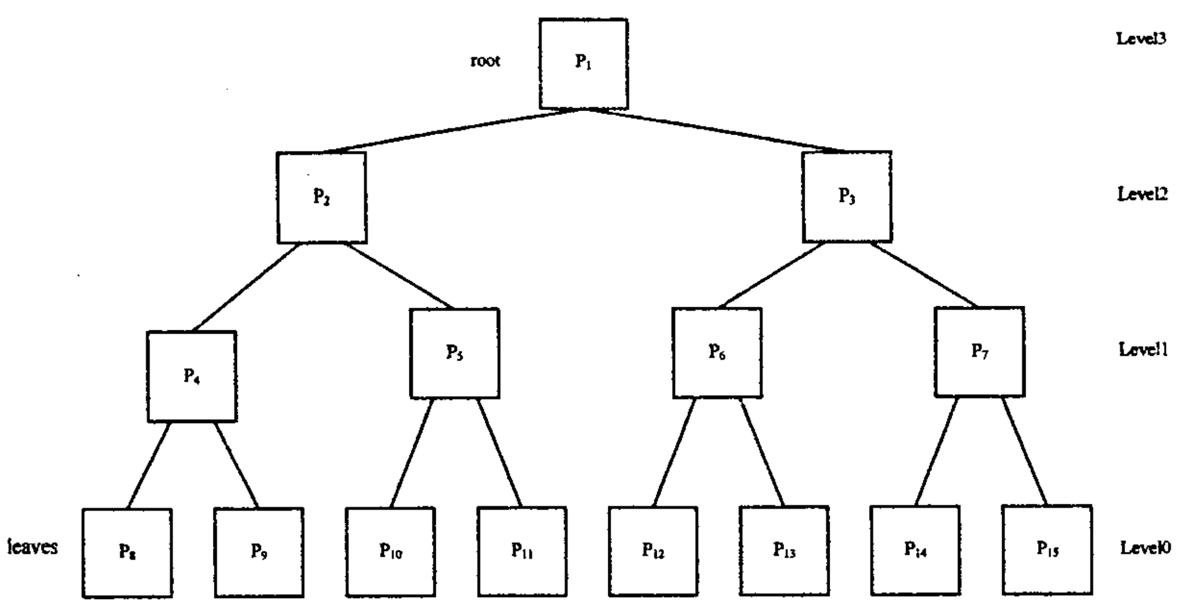


Figure 1.3 Tree Connection

(iv) Perfect Shuffle Connection:

Let N processors $p_0, p_1, \ldots, p_{N-1}$ be available where N is a power of 2. In the perfect shuffle interconnection a one- way line links pi to pj , where :

$$j = \begin{cases} 2i & \text{for } 0 \le i \le N/2-1, \\ 2i+1-N & \text{for } N/2 \le i \le N-1, \end{cases}$$

as shown in Fig. 1.4 for N=8.

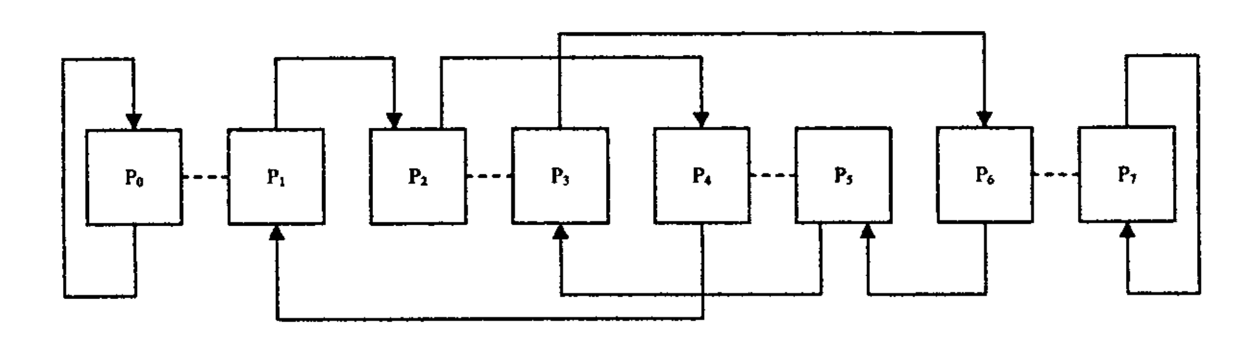


Figure 1.4 Perfect Shuffle Connection

(v) Cube Connection:

Assume that $N= 2^q$ for some $q \ge 1$ and let N processors be available $p_0, p_1, .., p_{N-1}$. A q-dimensional cube (or hypercube) is obtained by connecting

each processor to q neighbors. This is illustrated in Fig. 1.5 for q=3 . The indices of $p_0, p_1, .., p_7$ are given in binary notation.

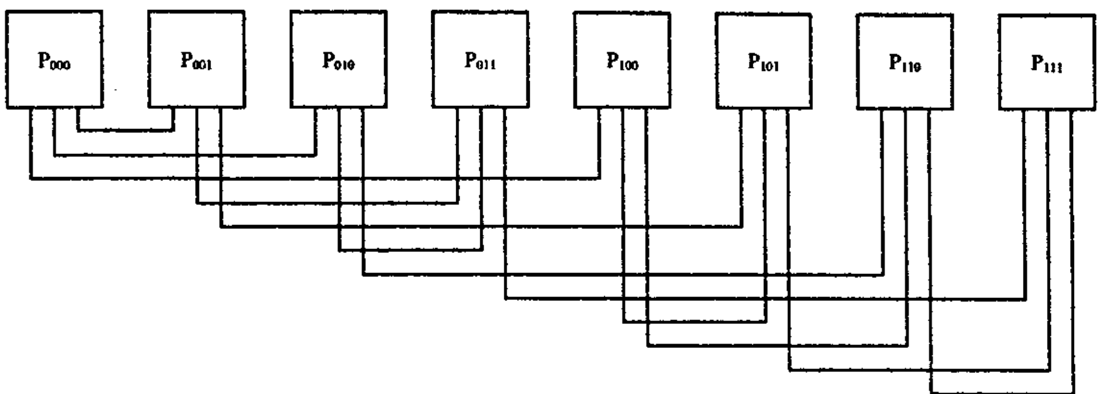


Figure 1.5 Cube Connection for N=8

Thesis of Parallel Computation:

[MJQu,1994] It is well known that space and time are the two most important resources characterizing the complexity of sequential computation. Time is still important in parallel computations: the main reason for studying parallel computation is the attempt to "speed up" the solution of computational problems, and with more processing units available the time needed to solve a given problem will be less. This fact points at processing units as the second main requirement for parallel computation.

Parallel Computations and Algorithmic Models:

The Algorithmic Model:

[MJQu,1994] The model is described by the following rules:

1- at any instant , any number of processors can be used;
2- every processor can execute any operation (arithmetical or logical) in a unit of time;
3- data access is at no cost;
4- communication among processors is at no cost.

The crucial resources needed by the model are the number of parallel steps (number of time units) and the number of processors used , as a function of problem size.

Boolean Circuits:

[MJQu,1994] , [JJAJ,1992] The ordinary computers are parallel machines. Each computer is , in fact, made of electronic circuits with millions of switching elements and transmission lines operating in parallel. Boolean circuits are a logic model of electronic circuits.

The model works on problems represented by sequences of bits. All information entering and exiting a real computer is encoded in a sequences represent data

with some sort of structure, the meaning of a sequence depends on interpretation.

Since operations are executed at the bit level, when an algorithm is expressed as a Boolean circuit its cost is called bit cost (while its complexity is called bit complexity).

This is a measure of parallel cost. In a sense, the above characteristics make the study of the complexity of parallel computations.

The Parallel Algorithm:

The parallel algorithm defines as a set of operations that each of it execut it's sequential algorithm and they are able to communicate among them selves by interconnection network.

A parallel algorithm is a solution method for a given problem destined to be performed on a parallel computer. In order to properly design such algorithms, one needs to have a clear understanding of the model of computation underlying the parallel computer.

Analyzing Parallel Algorithms:

[SeGA-1989] , [AnJe-1974] , [GSAG-1994] The analyzing parallel algorithms determines how good an algorithm is, how fast, how expensive to run, and how efficient it is in its use of the available resources. It is usually evaluated using the following criteria: running time, number of processors used, and cost.

1- Running Time:

This is defined as the time taken by the algorithm to solve a problem on a parallel computer, that is, the time elapsed from the moment the algorithm starts to the moment it terminates.

2- Number of processors:

The second most important criterion in evaluating a parallel algorithm is the number of processors it requires to solve a problem . it costs money to purchase, maintain, and run computers.

3- Cost:

The cost of a parallel algorithm is defined as the product of the previous two measures; hence

Cost = Parallel running time × Number of processors used

In other words , cost equals the number of steps executed collectively by all processors in solving a problem in the worst case. This definition assumes that all processors execute the same number of steps. If this is not the case, then cost is an upper bound on the total number of steps executed.

Designing Parallel Algorithms:

[IFos-1994] , [JeHk-1997] , [BrCMal-1992] , [JHKi-1990] The most natural approach to developing parallel algorithms is to start from a sequential

algorithm, then to identify the independent steps, and finally to obtain a parallel version.

This simple conversion from sequential to parallel often produces algorithms with a low degree of parallelism. Hence there arises the problem of determining strategies for obtaining efficient parallel algorithms. There is no general method for the best parallelization of a given problem. However, there are techniques that are successfully used in different problems. We will introduce a few widely applicable methods.

1- Divide et impera:

It consists of dividing the problem into subproblems of the same kind but of lesser size and then rearranging the solutions. This technique is sometimes very efficient. In fact, it allows us to express problems in terms of independent subproblems. These in turn can be solved in parallel, and their solution to the original problem.

2- Vectorization:

Vectorization is a technique that, starting from the analysis of a problem, produces algorithms working on data structures well suited for vector computations (typically vectors and matrices).

3- Vector iterations:

The method of vector iterations has been introduced by Traub and others for solving tridiagonal linear systems. Starting from a direct sequential method, a parallel iterative method is built.

4- Recursive doubling:

Recursive doubling consists of turning a computation graph such as the one shown in figure 2.1(a) into the graph shown in figure 2.1(b). it enables us to obtain a graph with logarithmic depth from a graph with linear depth.

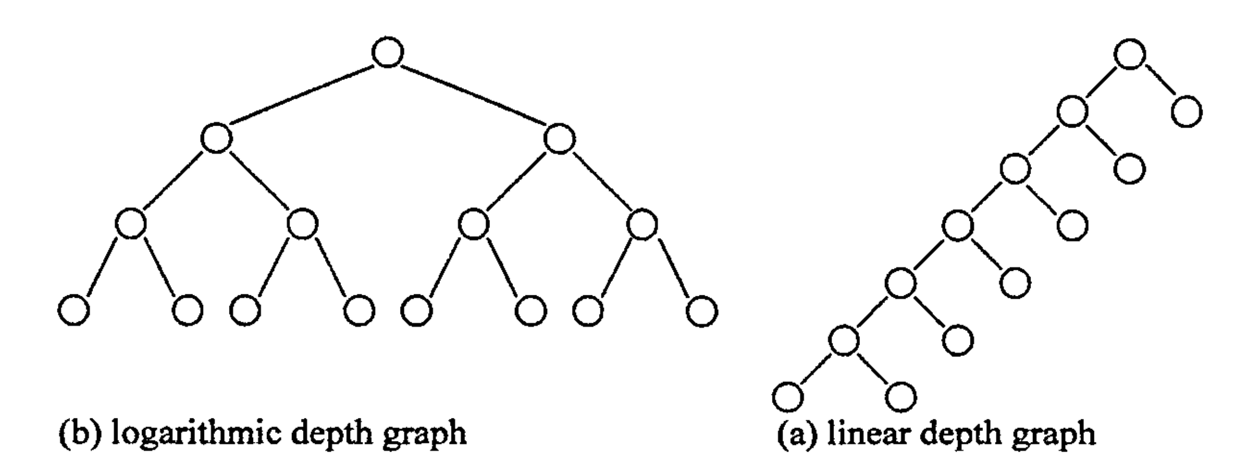


Figure 2.1

Reducing the number of processors:
Sometimes it is possible to reduce the order of magnitude of the number of processors used by a parallel algorithm by increasing the execution time by a constant factor.

The Algorithm Cost:

[WaCDaK] , [TLCP-1993] , [JHro-1997]The cost is defined as the product of the number of processors used in the algorithm and the running time of the algorithm.

The Comparison Of The Parallel Algorithms With The Sequential Algorithms:

[RaRz-1993] , [JeHk-1997] The analysis of the algorithms declares in the comparison of the algorithms that dissolve the same problem. For any machine and for any programming language:
Let us assume that we have two algorithms A and B. the two algorithms are executed the same problem. We want to know that the algorithm A is better than the algorithm B.

Parallel Algorithms To Solve Some Problems On Multiprocessors Computers:

Broadcast Algorithm:

Let D be a location in memory holding a datum that all N processors need at a given moment during the execution of an algorithm. This is a special case of the more general multiple-read situation and can be simulated on an EREW computer by broadcasting process. We gave this process formally as procedure BROADCAST.

The Supervision Of The Termination:

Sometimes we need to know whether the processors are done its work or not. A location F holding a Boolean value can be set aside in the shared memory to signal that one of the processors has terminated and, consequently, that all other processors should terminate their work. Initially , F is set to false when a processor terminate it's work it sets F to true. At every step of the work all processors check F to see if it is true and stop if this is the case. When we work on the EREW model this process log N steps are needed to broadcast the value of F each time the processors need it. This lead to increase the running time. In the CREW SM SIMD computer, since concurrent-read operations are allowed, it takes one step for all processors to read F and this decrease the running time. Finally we note that sometimes more than one processor terminate its work in

the same time and hence more than one processor may need to report success at the same time. This means that two or more processors will attempt to write into location F simultaneously, a situation that can only be handled by a CREW SM SIMD computer.

Searching On A Tree:

[GC.O,1993], [KMJm,1982] , [CSJJ,1981] Searching is one of the most fundamental operations in the field of computing. It is used in any application where we need to find out whether an element belongs to a list or, more generally, retrieve from a file information associated with that element. In its most basic form the searching problem is stated as follows: Given a sequence S = { $s_1, s_2, .., s_n$} of integers and an integer x, it is required to determine whether x = s_k for some s_k in S.

In sequential computing , the problem is solved by scanning the sequence S and comparing x with its successive elements until either an integer equal to x is found or the sequence is exhausted without success. A tree- connected SIMD computer with n leaves is available for searching a file of n records. Each leaf of the tree stores one record of the file to be searched. The root is in charge of receiving input from the outside world and passing a copy of it to each of its two children . it is also responsible for producing output received from its two children to outside world. As for the intermediate nodes, each of these is capable of:

1- receiving one input from its parent, making two copies of it, and sending one copy to each of its two children ; and
2- receiving two inputs from its children, combining them, end passing the result to its parent.

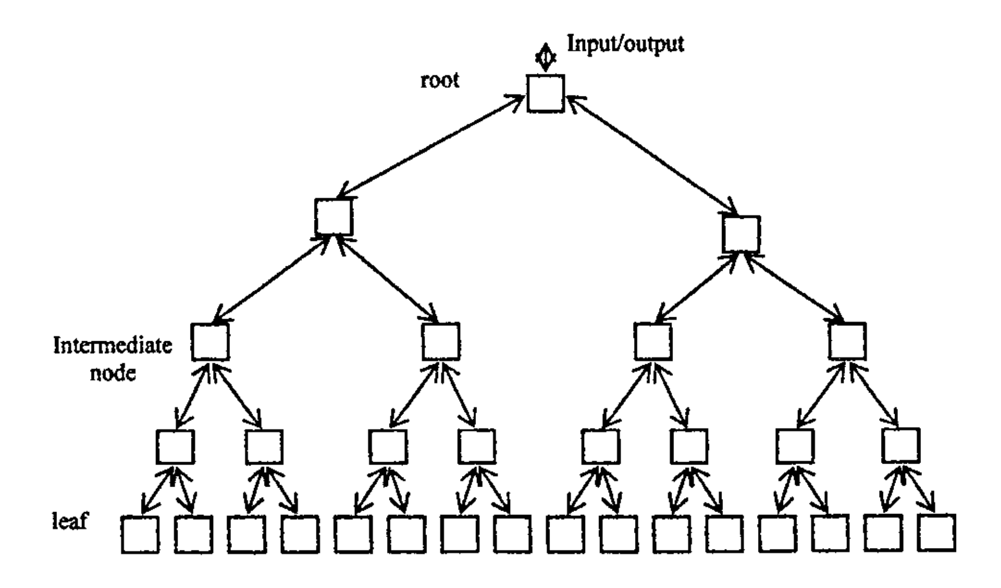


Figure 3.2 Tree-connected computer for searching

Parallel Algorithms Applications:

In this section it is shown some of decision and optimization problems applications that used efficient parallel algorithms such as computing prefix sums. We use the problems of job sequencing with deadlines which is decision problem and knapsack problem which is optimization one.

In the two problems there are an algorithm on parallel computer with tree-connection and each of them used computing prefix sums. Since it is shown several computer models.

- SEQUENTIAL SUMS (for the sequential case).
- PARALLEL SUMS (for multiprocessor computer).
- ALLSUMS (for computing all sums)
- Prefix sums of a tree
- Prefix sums of mesh

Job Sequencing with deadline:

A set of n jobs $J = \{j_0, j_1, .., j_{n-1}\}$ is given to be processed on a single machine. The machine can execute one job at a time , and when it is assigned a job, it must complete it before the next job can be processed. With each job j_1 is associated:

(i) a processing t_i and

(ii) a deadline d_i by which it must be completed.

A schedule is a permutation of the jobs in J that determines the order of their execution . a schedule is said to be feasible if each job finishes by its deadline. The question is: Given n jobs $\{j_0, j_1, .., j_{n-1}\}$ with processing time $\{t_0, t_1, .., t_{n-1}\}$ and deadlines $\{d_0, d_1, .., d_{n-1}\}$, does a feasible schedule exist? It turns out that this question can be answered in the affirmative if and only if any schedule where the jobs are executed in nondecreasing order of deadlines is feasible. Therefore , to solve the problem, it suffices to arrange the jobs in order of nondecreasing deadlines and test whether this yields a feasible schedule. In case it does , we know that the answer to the question is yes, otherwise the answer is no. sequentially ,this algorithm requires O(n log n) time to sort the jobs and then O(n) time to test whether each job can be completed by its deadline.

The Knapsack Problem:

[MLFi,1981], [WShi,1979] , [DFGP,1982]We are given a knapsack that can carry a maximum weight of W and a set n objects $A=\{a_0, a_1, .., a_{n-1}\}$ whose respective weights are $\{w_0, w_1, .., w_{n-1}\}$ associated with each object is a profit, the set of profits being denoted by $\{p_0, p_1, .., p_{n-1}\}$. If we place in the knapsack a function z_i of the object whose weight is w_i , where $0 \leq z_i \leq 1$, then a profit of $z_i p_i$ is gained. Our purpose is to fill the knapsack with objects (or fractions thereof) such that

(i) the total weight of the total weight of the selected objects does not exceed W and
(ii) the total profit gained is large as possible. We have dissolved this problem on parallel computer with tree- connection and on a mesh-connection parallel computer.

Parallel Algorithm On Multiprocessors Computer With Shared Memory:

Our study begins by showing some famous problems and enclose their definitions and its sequential and parallel algorithms for several models of computer and its connection-models. Since, it is shown an analysis of each algorithm for study its efficiency and its time cost with some examples.
We will have two problems of comparison problems. The first is known as the selection problem. It rises in many applications in computer science and statistics. The second is that of merging. Which belongs to comparison problems. In computer science, merging arises in a variety of contexts including database applications in particular and the file management in general. Many of these applications , of course, involve the merging of nonnumeric data. Furthermore, it is often necessary once the merging is complete to delete duplicate entries from the resulting sequence.

Selection Problem:

Our study of parallel algorithm design and analysis take in the following problem:
Given a sequence S of n elements and an integer k, where $1 \leq k \leq n$, it is required to determine the k th smallest element in S. this is known as the selection problem. It arises in many applications in computer science and statistics. Our purpose is to present a parallel algorithm for solving this problem on the shared-memory SIMD model. The algorithm will be designed to meet a number of goals, and our analysis will then confirm that these goals have indeed been met. This problem belongs to family of problems known as comparison problems. These problems are usually solved by comparing pairs of elements of an input sequence. We start in this section by the selection problem formally and deriving a lower bound on the number of steps required for solving it on a sequential computer. This translates into a lower bound on the cost of any parallel algorithm for selection. An optimal sequential algorithm is presented. Our design goals are stated in the form of properties generally desirable in any parallel algorithm. Since, there are the parallel selection algorithm and its analysis.

Merging problem:

This problem belongs to a class of problems known as comparison problems. It is defined as follows :

let $A=\{a_0,a_1,..,a_{n-1}\}$ and $B=\{b_0,b_1,..,b_{n-1}\}$ be two sequences of numbers sorted in non decreasing order ; it is required to merge A and B , that is, to form a third sequence $C = \{c_0,c_1,..,c_{r+s}\}$, also sorted in nondecreasing order, such that each c_i in C belongs to either A or B and each a_i and b_i appears exactly once in C.

A Network For Merging:

We saw that special-purpose parallel architectures can be obtained in any one of the following ways:

(i) using specialized processors together with a conventional interconnection network,
(ii) using a custom-designed interconnection network to link standard processors, or
(iii) using a combination of (i) and (ii).

We shall take the third of these approaches.

Merging will be accomplished by a collection of very simple processors communicating through a special-purpose network. This special-purpose parallel architecture is known as an (r,s)-merging network. All the processors to be used are identical and are called comparators. As illustrated by Fig 3.9.

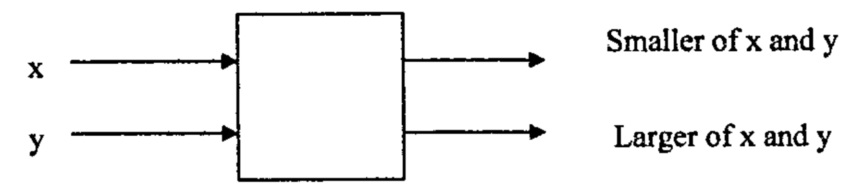


Figure 3.9 comparator

The only operation a comparator is capable of perfoming is to compare the values of its two inputs and then place the smaller and larger of the two on its top and bottom output lines, respectively.

We begin by describing a special-purpose parallel architecture for merging. A parallel algorithm for the CREW SM SIMD model is presented that is adaptive and cost optimal. Since the algorithm invokes a sequential procedure for merging , that procedure is also described. It is shown how the concurrent- read operations can be removed from the parallel algorithm by simulating it on an EREW computer. Finally, an adaptive and optimal algorithm for the EREW SM SIMD model is presented whose running time is smaller than that of the simulation. The algorithm is based on a sequential procedure for finding the median of two sorted sequences also described.

The Results:

- The study of the parallel algorithms this new subclass of the informatics subclasses is a difficult problem in the actual time especially because there isn't acceptable machines that allow off compiler the given algorithms in this study to have practical programs and when we execution it on factual problems we materialize the pith of the results that possibly we access it.
- We have explained in the first chapter the intension of the multiprocessors systems and their definition, classification, and the study of how their connection. As we have introduced results and ideas about the theory of parallel computation, with some models of parallel computation.
- We have studied in the second chapter the parallel algorithms and their kinds, properties, design techniques, analysis measures, and cost limitation.
- We have introduced in the third chapter a crucial problems and their parallel algorithms on multiprocessors computers. The parallel algorithms help in recurrence more information and best solution in decreasing the cost of the problems. This problems give liberally to illustration the touch of the parallelism in increasing the size of problems that can solve and decreasing the optimal solution that is the algorithm of supervision of the termination reports an early access to the solution, thus, conducts to decreasing the running time.

The Proposals And The Recommendations:

- I suggest proceeding gross study for all the problems in the parallel system, and understanding the methods of parallel computers and their connection networks for allocation the suitable algorithms for solve this problems.
- Essential proceeding profound studies for destination the processes to the processors to have the best results from the parallel programming.
- Developing the ancient algorithms to solve the problems that there are not gross solution for them.
- Putting a through study for the transport of the information problem and search for algorithms that allow in treatment, change, and developing them.

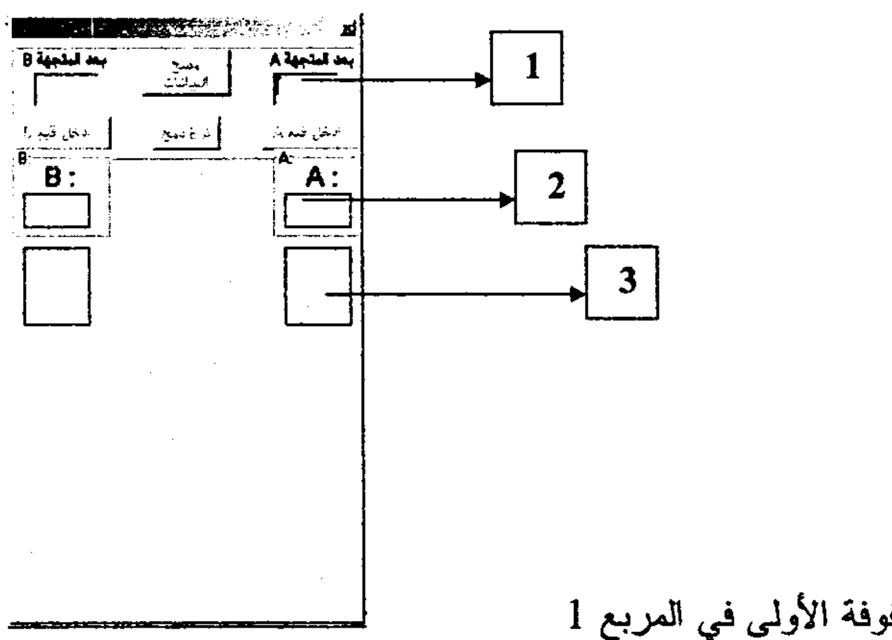
References:

- [SeGA] - Selim G.Akl " The Design & Analysis Of Parallel Algorithms " – prentice-Hall international, inc – 1989
- [MJQu] - M.J.Quinn " Parallel Computing , Theory and Practice" McGraw-Hill , Series in Computer Science , Second edition,1994
- [DFGP] – D. fayrd; G. Plateau " An Algorithm For The Solution Of The 0-1 Knapsak Problem" – Computing 28 - 1982
- [WShi] – W. Shih "A Branch And Bound Method For The Multi Constraint 0-1 Knasack Problem" - Journal of The Operational Research Society, 30(4) – 1979
- [MLFi] – M.L.Fisher "The Lagangian Relaxation Method Of Solving The Multiconstraint 0-1 Knapsack To Optimality" – Managemment Science 27 -1981
- [CSJJ] – C. Savage ; J.Ja'Ja " Fast Efficient Parallel Algorithms For some Graph problems" – SIAM J. On Computing, vol.16 n⁰4,Nov.1981
- [KMJm] – K.M.Chndy, J.Misra "Distributed Computation On Graphs :Shortest Path Algorithm" – CaCm , vol 25, n⁰ 11,Nov.1982
- [GC.O] - Gary Chartrand & Ortrud R.Oellermann – "APPLIED AND ALGORITHMIC GRAPH THEORY" – McGraw-Hill,Inc – 1993
- [WaCDaK]– Ward Cheney,David Kincaid " NUMERICAL MATHEMATICS AND COMPUTING " – Brouks/cole publishing company.
- [TLCP] – T.L.Freeman , C.Philips " Parallel Numerical Algorithm" - Prentice Hall – 1993
- [JHro] – J.Hromkovi "Communcation Complexity And Parallel Computing ; The Application Of Communication Comlexity In Parallel Computing" – Springerverleg , Berlin and Heidelberg GmbH & Co.KG,Feb.1997
- [IFos] - I.Foster " Designing And Building Parallel Programs " – Addison-Wesley – 1994
- [JeHk] – Jeffery H Kingston "Algorithms and Data structures" – Addison Wesley – 1997
- [BrCM] – Brouno Codenotti,Mauro Leoncini "Introduction to Parallel Processing" – Addison Wesley Publishing company - 1992
- [JHKi] – J.H.Kingston " Algorithms And Data Structures , Design, correctness , Analysis" – Addison-Wesley - 1990
- [AnJe] - A.N.Aho , J.E.Hopcroft and j.D.Ulman "The Design And Analaysis Of computer Algorithms" - Addison Wesley - 1974
- [RaRz] – الدكتور راكان رزوق "الخوارزميات وبنىالمعطيات" – منشورات المعهد العالي للعلوم التطبيقية والتكنولوجيا ، قسم المعلوماتية - 1993

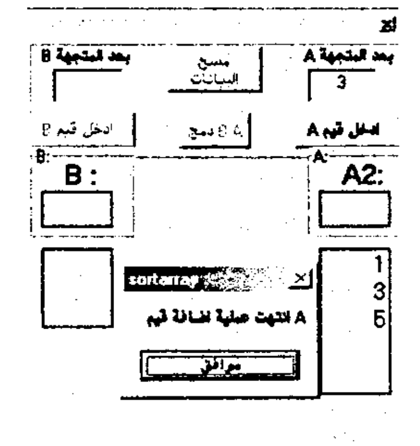
ملحق

المساعد لبرنامج دمج مصفوفتين

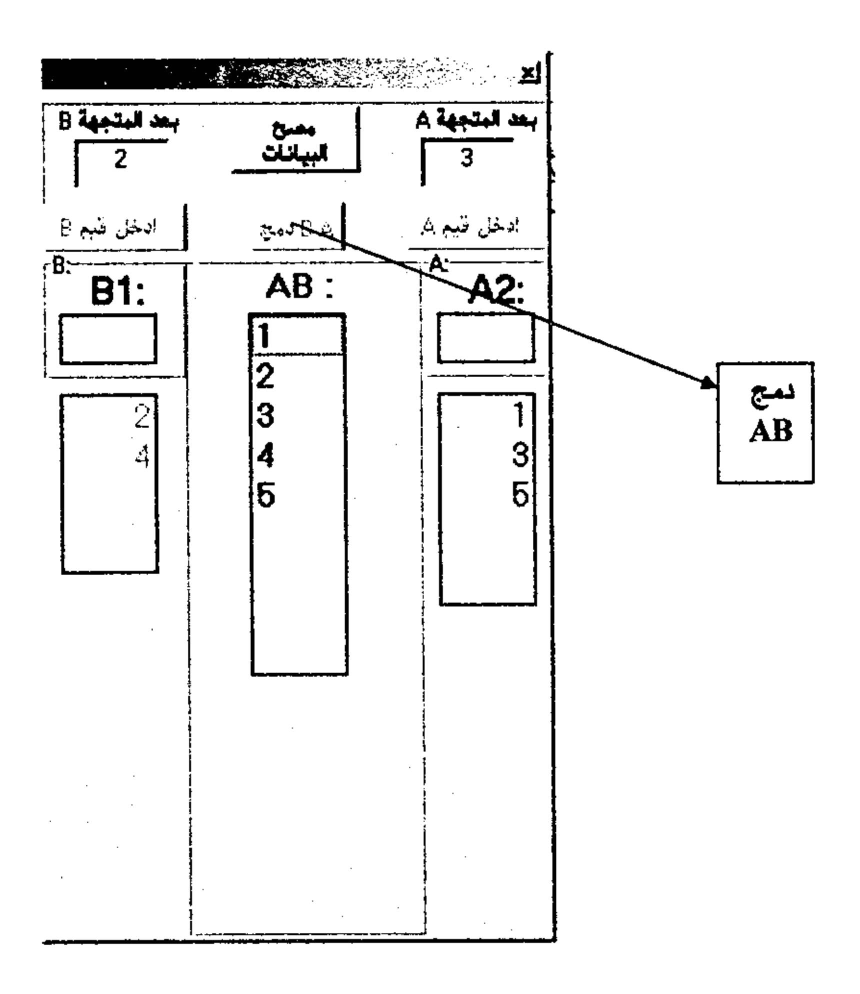
آلية استخدام البرنامج:



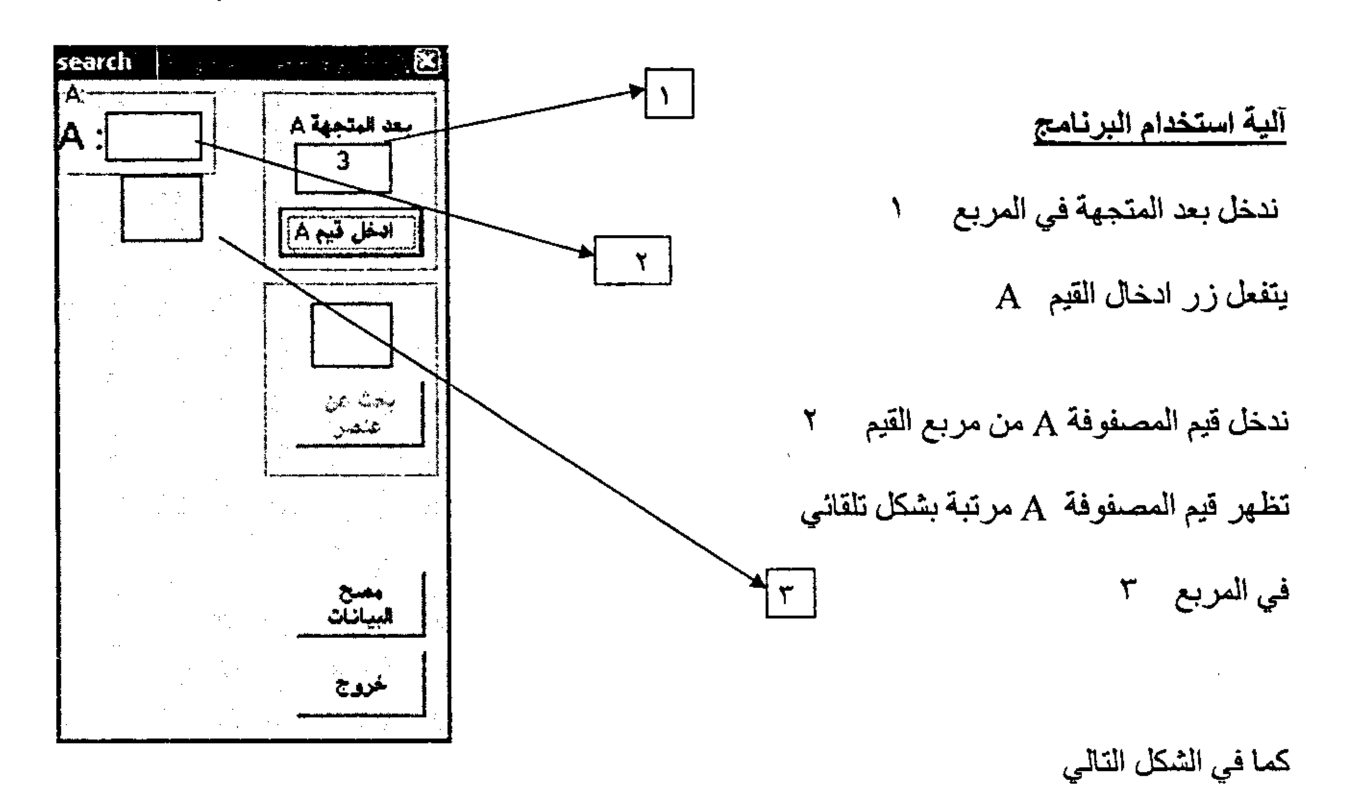
- ندخل بعد المصفوفة الأولى في المربع 1
- يتفعل زر إدخال قيم A
- ندخل قيم المصفوفة A في المربع 2
- تظهر قيم المصفوفة A مرتبة بشكل تلقائي في المربع 3 كما في الشـكل التـالي:



ويتم إدخال قيم المصفوفة B بطريقة مماثلة، بعد ذلك يتم دمج المصفوفتين في مصـــفوفة جديدة بالضغط على الزر دمج AB. وبعد الانتهاء يمكن تفريغ البيانات باســتخدام الــزر مسح البيانات وبالتالي يعود البرنامج للوضع الذي يسمح بتكرار ما سبق.



المساعد لبرنامج البحث في مصفوفة



<u>آلية استخدام البرنامج</u>

ندخل بعد المتجهة في المربع ١

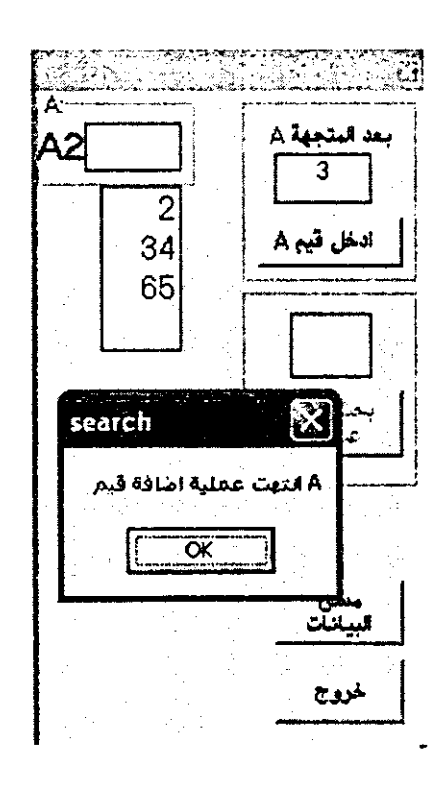
يتفعل زر ادخال القيم A

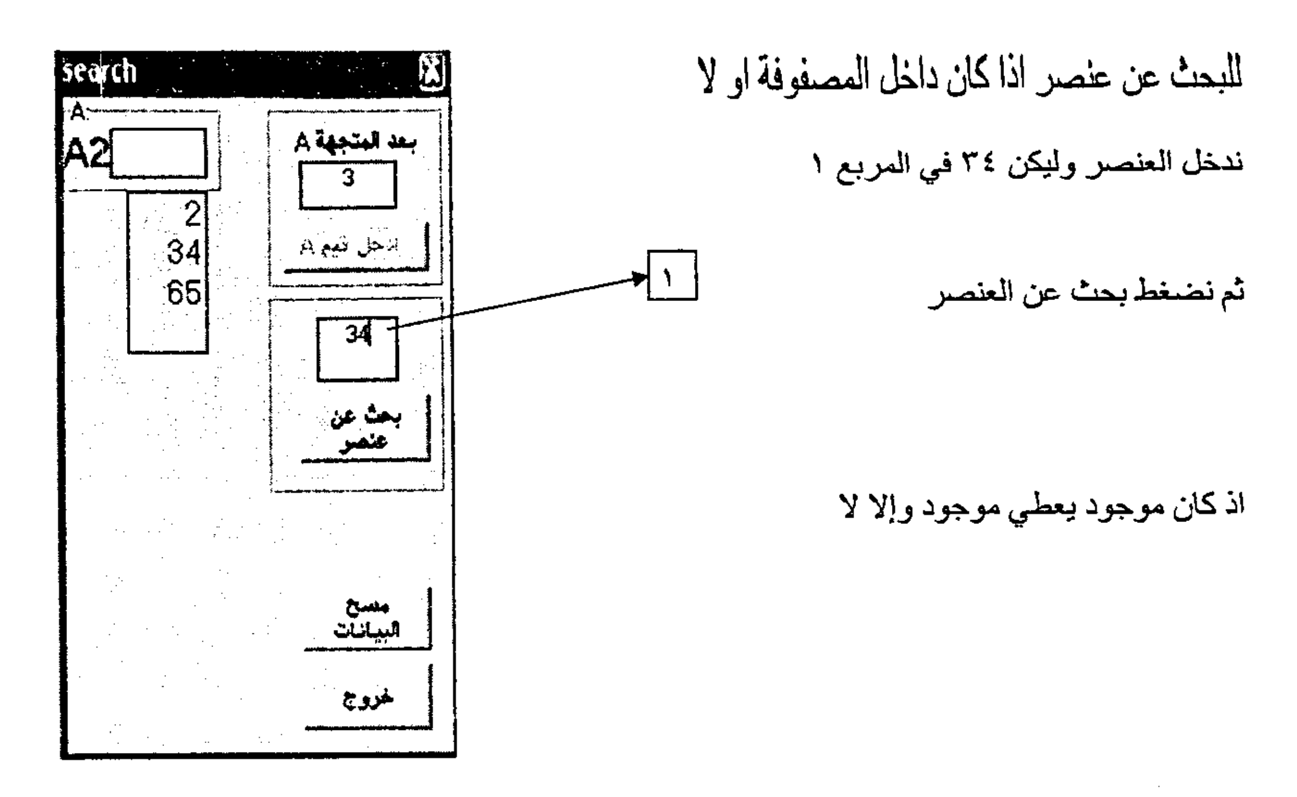
ندخل قيم المصفوفة A من مربع القيم ٢

تظهر قيم المصفوفة A مرتبة بشكل تلقائي

في المربع ٣

كما في الشكل التالي



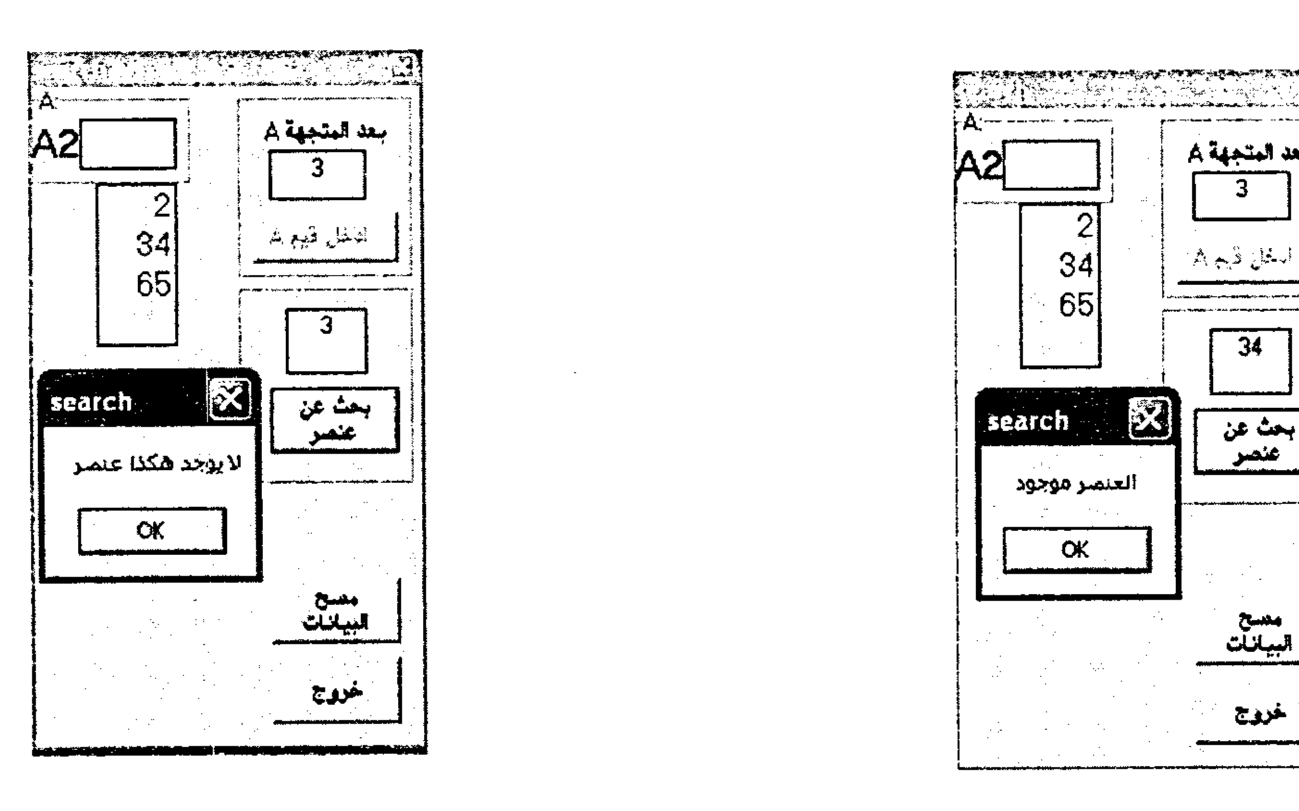


للبحث عن عنصر اذا كان داخل المصفوفة او لا

ندخل العنصر وليكن ٣٤ في المربع ١

ثم نضغط بحث عن العنصر

اذ كان موجود يعطي موجود وإلا لا



تتم عملية البحث باستخدام خوارزمية تعاودية ، تكرار تقسيم المصفوفة الى ان تصبح المصفوفة عنصر واحد هي ذالك العنصر ان وجد
كما يلي

```
CForm1::divary(  CVBArray<int>   &ar, int &d, int &X)
     {

          CVariant Ti ;
CVariant Tj ;
CVariant TN ;
BYTE TM = 0;
 CVBArray<int>TA (CVBBounds(50 ), CVBBounds())  ;
CVariant vbOKOnly ;



               Tj = 0;
       TN = d;
      if    ( d == 0 ) {
                         MsgBox("لا يوجد هكذا عنصر");
return;

                  }

         if        ( X == 0 & ar ( 0) != 0 ) {
                         MsgBox("لا يوجد هكذا عنصر");
return;

                  }

         TM = d % 2;
      if    ( TM == 1 ) {
                            TM = d / 2 + 1;
      if    ( ar ( TM) == X ) {
                         MsgBox(" العنصر موجود");
return;

                  }
       else       {
                            d = d / 2;
      if    ( X < ar ( d) ) {
                           for (Ti=0; Ti<=(CVariant)(d - 1); Ti++) {
                     TA ( Tj) = ar ( Ti);
       Tj = Tj + (CVariant)(1);

             }

divary ( TempLocn ( TA), d, X);

                  }
       else if    ( X == ar ( d) ) {
                         MsgBox(" العنصر موجود");
return;

                  }
       else if    ( X > ar ( d) ) {
                           for (Ti=d + 1; Ti<=TN; Ti++) {
                     TA ( Tj) = ar ( Ti);
       Tj = Tj + (CVariant)(1);

             }

divary ( TempLocn ( TA), d, X);

                  }
```

```
        else        {
                            MsgBox("موجود العنصر");
return;

                    }

                    }

                    }
        else        {
                              d = d / 2;
      if     ( X < ar ( d) ) {
                            for (Ti=0; Ti<=(CVariant)(d - 1); Ti++) {
                        TA ( Tj) = ar ( Ti);
        Tj = Tj + (CVariant)(1);

                }

divary ( TempLocn ( TA), d, X);

                    }
        else if     ( X == ar ( d) ) {
                            MsgBox("موجود العنصر");

                    }
        else if     ( X > ar ( d) ) {
                            for (Ti=d + 1; Ti<=TN; Ti++) {
                        TA ( Tj) = ar ( Ti);
        Tj = Tj + (CVariant)(1);

                }

divary ( TempLocn ( TA), d, X);

                    }
        else        {
                            MsgBox("موجود العنصر");

                    }

                    }

      }
```